

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開平11-149385

(43)公開日 平成11年(1999)6月2日

(51) Int.Cl.⁶

G O 6 F 9/46

識別記号

350

340

FI

G O 6 F 9/46

350

340A

審査請求 未請求 請求項の数20 O.L (全 29 頁)

(21)出願番号 特願平10-8299

(22)出願日 平成10年(1998)1月20日

(31)優先權主張番号 特願平9-248178

(32)優先日 平9(1997)9月12日

(33) 優先権主張国 日本 (J P)

(71)出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(72) 發明者 関口 知紀

神奈川県川崎市麻生区王禅寺1099番地 株

式会社日立製作所システム開発研究所内

(72) 発明者 新井 利明

神奈川県川崎市麻生区王禅寺1099番地 株

式会社日立製作所システム開発研究所内

(72) 發明者 金子 茂則

茨城県日立市大みか町五丁目2番1号 株

式会社日立製作所大みか工場内

(74) 代理人 弁理士 小川 勝男

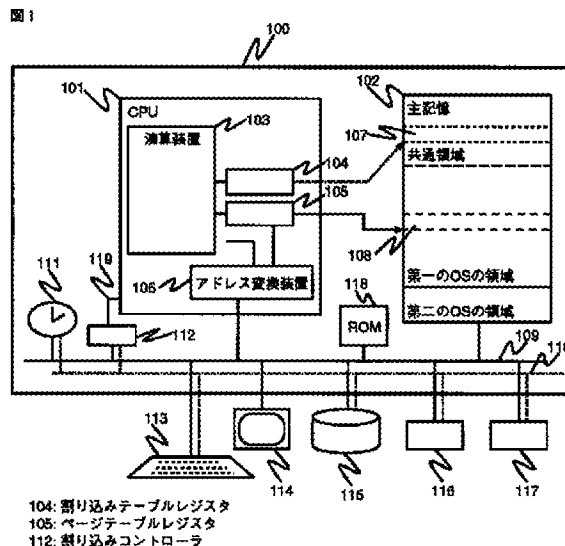
最終頁に続く

(54)【発明の名称】 マルチOS構成方法

(57) 【要約】

【課題】本発明は、従来の仮想計算機方式の複雑でオーバヘッドの大きいOS制御方式に対し、ソフトウェアだけで容易に一台の計算機で複数OSを同時走行する方式を提供する。本発明によれば、第一のOSから完全に独立して動作する新機能を完全に計算機に組み込むことができ、それにより割込応答時間の短縮や信頼性の向上を実現できる。

【解決手段】第一のOSと他のOSが管理するハードウェア資源を分割する手順と、他のOSを起動する手順と、実行OSを切替える手順と、割り込み要因により割り込み処理するOSを決定し、適切な割り込みハンドラを起動する手順により、複数のOSの同時実行を実現する。



【特許請求の範囲】

【請求項1】 計算機の制御方法に関し、第一のオペレーティングシステムの起動時に他のオペレーティングシステムが利用するハードウェア資源を第一のOSの管理対象から外して他のオペレーティングシステムに与える手順と、すべてのオペレーティングシステムで共有している領域に配置した共通の割り込み処理手順とを有し、前記割り込み処理手順が各オペレーティングシステムの実行をスケジュールすることにより、各オペレーティングシステムが実行する特権命令のエミュレート処理、および、装置なしで、複数のオペレーティングシステムを一台の計算機で同時に動作させることを特徴とするマルチOS構成方法。

【請求項2】 請求項1のマルチOS構成方法であって、第一のオペレーティングシステムの初期化処理時に、第二のオペレーティングシステム用のハードウェア資源を第一のオペレーティングシステムの初期化以降の処理ではアクセスできないように予約する手順と、予約した物理メモリに第二のオペレーティングシステムをロードして第一のオペレーティングシステムとは別の仮想アドレス空間で第二のオペレーティングシステムを起動する手順と、第一のオペレーティングシステムのカーネル領域の一部分を第二のオペレーティングシステムと共有するように設定する手順と、第二のオペレーティングシステムが受け取る外部割り込みについて、第一のオペレーティングシステムからは割り込み禁止にできないように第一のオペレーティングシステムのデータ構造を変更する手順とを有することを特徴とするマルチOS構成方法。

【請求項3】 請求項1のマルチOS構成方法であって、割り込みを捕獲したときに割り込み要因よりどのオペレーティングシステムに割り込み処理させるかを決定する手順と、第一のオペレーティングシステムから第二のオペレーティングシステムのモジュールを呼び出す手順と、第二のオペレーティングシステムの処理が終了したときに第一のオペレーティングシステムに復帰する手順を有することにより、2つのオペレーティングシステムを一台の計算機上で同時に動作させることを特徴とするマルチOS構成方法。

【請求項4】 請求項1ないし3のマルチOS構成方法であって、第一のオペレーティングシステムが回復不可能な障害で停止したときに第二のオペレーティングシステムにそのことを通知する手順と、停止時に第二のオペレーティングシステムが管理しているデバイスの割り込みを許可する手順と、第二のオペレーティングシステムの処理終了時に、第一のオペレーティングシステムが停止しているかを判定し、停止している場合は割り込み待ちを実施する手順を有することにより、第一のオペレーティングシステムが停止しても第二のオペレーティングシステムが動作を継続できることを特徴とするマルチOS構成方法。

【請求項5】 請求項1ないし4のマルチOS構成方法であって、両方のオペレーティングシステムで共有しなければならない命令コードとデータを専用の領域に閉じ込めたカーネルオブジェクトファイルを利用し、第二のオペレーティングシステム起動時に第一のカーネルオブジェクトファイルの前記領域を見つけ、その領域のみを共有領域に設定する手順を有することで、共有領域を小さくできることを特徴とするマルチOS構成方法。

【請求項6】 請求項1ないし5のマルチOS構成方法であって、第二のオペレーティングシステムの実行中に第一のオペレーティングシステムの管理するデバイスからの割り込みを受けたときにはそれを記録する手順、あるいは、装置と、第二のオペレーティングシステムの処理が終了して第一のオペレーティングシステムに制御を戻す時に、第二のオペレーティングシステム実行中に発生した割り込みの処理を開始する手順と、第二のオペレーティングシステムが管理しているデバイスからの割り込みは、第一のオペレーティングシステムが実行中であっても即座にオペレーティングシステムを切り替えて割り込み処理を開始する手順とを有し、これらの手順により第一のオペレーティングシステムを、第二のオペレーティングシステムのアイドル時だけ動作するようにスケジュールして、第二のオペレーティングシステムが管理するデバイスが発生する割り込みに対する応答時間を短縮することを特徴とするマルチOS構成方法。

【請求項7】 請求項1ないし5のマルチOS構成方法であって、複数のプロセッサと、外部機器からの割り込みを特定のプロセッサ、あるいは、プロセッサ群に通知することを指定できる装置を持っている計算機で、第一のオペレーティングシステムの初期化時に第二のオペレーティングシステム用にプロセッサを予約する手順と、第二のオペレーティングシステムを前記手順で予約したプロセッサで起動する手順と、外部機器からの割り込みをそれぞれの機器を管理するオペレーティングシステムが走行しているプロセッサ、あるいは、プロセッサ群に通知するように設定する手順を有することを特徴とするマルチOS構成方法。

【請求項8】 計算機の制御方法に関し、第一のオペレーティングシステムが起動時に読み込まれる複数のファイルから構成されていて、ハードウェア依存の処理がカーネル本体のファイルから分離されている場合に、前記ハードウェア依存処理ファイルを変更して、請求項1ないし7を実現することを特徴とするマルチOS構成方法。

【請求項9】 請求項1のマルチOS構成方法であって、第一のオペレーティングシステムの初期化処理時に、他の複数のオペレーティングシステム用のハードウェア資源を第一のオペレーティングシステムの初期化以降の処理ではアクセスできないように予約する手順と、予約した物理メモリに他の複数のオペレーティングシステムをロードして第一のオペレーティングシステムとは別の仮

想アドレス空間で他のオペレーティングシステムを起動する手順と、第一のオペレーティングシステムのカーネル領域の一部分を他の複数のオペレーティングシステムと共有するように設定する手順とを有することにより、1プロセッサの計算機で複数のオペレーティングシステムが動作可能であることを特徴とするマルチOS構成方法。

【請求項10】請求項9のマルチOS構成方法であって、第一のオペレーティングシステムの内の他の複数のオペレーティングシステムと共有している領域にあるモジュールを介して、実行中のオペレーティングシステム以外のオペレーティングシステムのモジュールを呼び出す手順を有することを特徴とするマルチOS構成方法。

【請求項11】請求項9ないし10のマルチOS構成方法であって、割り込みを捕獲したときに割り込み要因より複数実行されているオペレーティングシステムのうちのどのオペレーティングシステムに割り込み処理させるかを決定する手順と、前記手順により決定したオペレーティングシステムの割り込み処理モジュールを呼び出す手順と、割り込み発生時に実行していたオペレーティングシステムに制御を戻す手順を有することにより、1つのプロセッサを有する計算機で複数のオペレーティングシステムを同時に動作させることを特徴とするマルチOS構成方法。

【請求項12】請求項9ないし10のマルチOS構成方法であって、1つ以上のオペレーティングシステムが回復不可能な障害で停止したときに、他の停止していない複数のオペレーティングシステムにそのことを通知する手順と、停止しているオペレーティングシステムが管理している割り込みを禁止する手順と、停止したオペレーティングシステムのモジュール呼び出しを禁止する手順を有することにより、停止した以外の複数のオペレーティングシステムが動作を継続できることを特徴とするマルチOS構成方法。

【請求項13】請求項9ないし10のマルチOS構成方法であって、複数のオペレーティングシステムで共有しなければならない命令コードとデータを専用の領域に閉じ込めたオブジェクトファイルを利用し、第一のオペレーティングシステム以外のオペレーティングシステム起動時に第一のオペレーティングシステムのオブジェクトファイルの前記領域を見つけ、その領域のみを共有領域に設定する手順を有することで、共有領域を小さくできることを特徴とするマルチOS構成方法。

【請求項14】請求項9ないし10のマルチOS構成方法であって、複数のオペレーティングシステムの間に実行優先度を設定する手順を有することを特徴とするマルチOS構成方法。

【請求項15】請求項14のマルチOS構成方法であって、実行中のオペレーティングシステムが他のオペレーティングシステムのモジュールを呼び出す時に、実行中

のオペレーティングシステムの優先度が呼出先のオペレーティングシステムの優先度よりも低ければ、即座に実行オペレーティングシステムを切替えてモジュール呼び出しを実施する手順と、実行中のオペレーティングシステムの優先度が呼出先のオペレーティングシステムの優先度よりも高ければ、モジュール呼び出し要求があることを記録して呼び出しを延期し、呼出先のオペレーティングシステムよりも優先度の高いオペレーティングシステムの処理が終了して、呼出先のオペレーティングシステムが実行可能になった時にモジュール呼び出しを実施する手順を有することを特徴とするマルチOS構成方法。

【請求項16】請求項14のマルチOS構成方法であって、実行中のオペレーティングシステムAの優先度よりも低い優先度のオペレーティングシステムBが処理する割り込みが発生した時にそれを記録して処理を延期する手順、あるいは、装置と、オペレーティングシステムAの処理が終了してオペレーティングシステムBが実行される時に、オペレーティングシステムA実行中に発生した割り込みの処理を開始する手順と、実行中のオペレーティングシステムよりも優先度の高いオペレーティングシステムが管理しているデバイスからの割り込みは、即座に実行オペレーティングシステムを切り替えて割り込み処理を開始する手順とを有し、優先度の高いオペレーティングシステムが管理するデバイスが発生する割り込みに対する応答時間を短縮することを特徴とするマルチOS構成方法。

【請求項17】請求項16のマルチOS構成方法であって、各オペレーティングシステムが管理する割り込みと、各オペレーティングシステムに割り当てられている優先度と、実行中のオペレーティングシステムに於いて、外部機器からの割り込みの許可と禁止の設定をする手段を有することを特徴とするマルチOS構成方法。

【請求項18】請求項9ないし17のマルチOS構成方法であって、複数のプロセッサと、外部機器からの割り込みを特定のプロセッサ、あるいは、プロセッサ群に通知することを指定できる装置を持っている計算機で、第一のオペレーティングシステムの初期化時に他の複数のオペレーティングシステム用にプロセッサを予約する手順と、他の複数のオペレーティングシステムを前記手順で予約したプロセッサで起動する手順と、外部機器からの割り込みをそれぞれの機器を管理するオペレーティングシステムが走行しているプロセッサ、あるいは、プロセッサ群に通知するように設定する手順を有することを特徴とするマルチOS構成方法。

【請求項19】請求項9のマルチOS構成方法であって、第一のオペレーティングシステム以外のオペレーティングシステムが受け取る外部割り込みについて、第一のオペレーティングシステムからは前記割り込みを禁止にできないように第一のオペレーティングシステムのデ

ータ構造を変更する手順とを有することを特徴とするマルチOS構成方法。

【請求項 20】請求項 1 ないし 6、および、請求項 8 ないし 17 のマルチOS構成方法であって、1つのプロセッサで複数のオペレーティングシステムを動作させることを特徴とするマルチOS構成方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は一台の計算機上で複数のオペレーティングシステムを稼働させるマルチOS構成方法に関する。

【0002】

【従来の技術】通常の計算機では1つのオペレーティングシステムが動作し、それが計算機のプロセッサ、メモリ、および、二次記憶装置等の計算機資源を管理し、計算機が効率良く動作できるように資源スケジュールを実施している。オペレーティングシステムには様々な種類がある。バッチ処理に優れるものや、TSS(Time Sharing System)に優れるもの、GUI(Graphical User Interface)に優れているものなど様々である。

【0003】一方で、これら複数あるオペレーティングシステムを1台の計算機で同時に実行したいというニーズがある。例えば、大型計算機においては、実際の業務に伴うオンライン処理を実行するオペレーティングシステムと、開発用のオペレーティングシステムを一台の計算機で動作させたいという要求がある。あるいは、GUIの整っているオペレーティングシステムと、実時間性に優れているオペレーティングシステムを同時に稼働させたい等という要求もある。

【0004】しかしながら、個々のオペレーティングシステムは、単独で計算機資源の管理を実施することを仮定しており、複数のオペレーティングシステムの共存は、何らかの機構なしには不可能である。

【0005】一台の計算機上で複数のオペレーティングシステムを動作させる機構としては、大型計算機で実現されている仮想計算機方式(OSシリーズ第11巻 V M、岡崎 世雄他著、共立出版株式会社)がある。仮想計算機方式では、仮想計算機制御プログラムが全ハードウェア資源を占有して管理し、それを仮想化して仮想計算機を構成する。仮想計算機を構成する制御部は、物理メモリ、入出力機器装置、外部割り込み等を仮想化する。

【0006】例えば、分割された物理メモリは、各仮想計算機に対してはあたかも0番地から始まる物理メモリのように振舞い、入出力装置を識別する装置番号も同様に仮想化されている。更に、磁気ディスクの記憶領域も分割して磁気ディスク装置の仮想化まで実現している。

【0007】各オペレーティングシステムは、制御プログラムにより構築された仮想計算機上で実行されるように制御プログラムによりスケジュールされる。しか

し、大型計算機における仮想計算機方式では、計算機資源を完全に仮想化、および、分割しようとするため、仮想計算機を構成する制御部分が複雑であり問題である。

【0008】また、特別なハードウェア支援がない場合、仮想計算機上で動作するオペレーティングシステムが発行する制御レジスタの設定、入出力命令等の特権命令は、仮想計算機制御プログラムによりエミュレートしなければならないため、オーバーヘッドが大きくなり問題である。実際、仮想計算機を実装している大型計算機では、仮想計算機用に特別なプロセッサ機能やマイクロコード等のハードウェアを追加してオーバーヘッドの削減を図っている。仮想計算機方式は、完全に計算機資源を仮想化することを目的としているため複雑であり、更に、仮想計算機の高性能化のためには特殊なハードウェア機構が必要であり問題である。

【0009】一方、一台の計算機で複数のオペレーティングシステムのインターフェイスを提供する技術として、マイクロカーネルがある。マイクロカーネルでは、マイクロカーネルの上に、ユーザに見せるオペレーティングシステム機能を提供するオペレーティングシステムサーバを構築し、ユーザはそのサーバを経由して計算機資源を利用する。オペレーティングシステム毎のサーバを用意すれば、ユーザに様々なオペレーティングシステム環境を提供することができる。

【0010】しかし、マイクロカーネル方式では、オペレーティングシステムサーバをマイクロカーネルに合わせて新規に構築する必要がある。多くの場合、現在あるオペレーティングシステムをマイクロカーネル上で動作するように変更することになるが、スケジューリング、メモリ管理等のカーネルの中核部分も変更することになり、変更箇所が多く、また、変更箇所がオペレーティングシステムの中核部分に及ぶため変更作業が複雑で容易でなく問題である。

【0011】また、オペレーティングシステムサーバはマイクロカーネルのサービスを利用することになるが、これは通常のオペレーティングシステムではないことであり、オーバーヘッドとなり性能低下をもたらす。

【0012】

【発明が解決しようとする課題】従来の仮想計算機方式は、複数のオペレーティングシステムを同時に動作させるために、全計算機資源を仮想化する手法によっていた。しかし、この方式では制御プログラムが複雑になる問題がある。更に、この方式では特権命令のエミュレーションが必要となるため、性能を得るには特別なハードウェアが必要であり問題である。

【0013】本発明は、オペレーティングシステムの初期化処理部分、割り込み管理部分の変更、および、割り込み管理プログラムの追加により、特別なハードウェアなしで複数オペレーティングシステムの同時実行を実現する。本発明では、特権命令のエミュレーションは不要

なため、各オペレーティングシステムの実行に新たなオーバーヘッドは伴わない。

【0014】本発明によれば、第一のオペレーティングシステムを補完する機能を容易に追加でき、高機能な計算機システムの構築が可能になる。更に、デバイスドライバとは異なり、第一のオペレーティングシステムとは全く独立して動作する機能を組み込むことができるため、第一のオペレーティングシステムに依存しない高信頼化機能を追加することも可能になる。

【0015】また、マイクロカーネル方式で複数のマルチオペレーティングシステム環境を構成する方法では、それぞれのオペレーティングシステムのインターフェイスを提供するオペレーティングシステムサーバの構築が難しいという問題がある。本発明によれば、オペレーティングシステムへの変更は初期化部分と割り込み管理部分のみに限定されるため、簡単にマルチオペレーティングシステム環境を構成できる。

【0016】

【課題を解決するための手段】本発明は、第一のOSにおいて、第二のOSが必要とする物理メモリ、外部デバイス等の計算機資源を予約し、どちらのOSからも独立した管理プログラムが外部割り込みを横取りして、割り込み要因によりどのOSの割り込みハンドラを起動すべきか決定し、OSの実行状態により割り込みハンドラを起動するタイミングを決定して、その基づいて各OSの割り込みハンドラを起動することにより、2つのOSを一台の計算機で動作させる。

【0017】

【発明の実施の形態】本発明の実施の形態について説明する。

【0018】以下、図面を用いて本発明の実施の形態について説明する。図1は本発明の実施の形態における計算機100の構成を示す図である。

【0019】計算機100は、プロセッサ101、主記憶装置102、バス109、割り込み信号線110、クロック割り込み生成器111、割り込み制御装置112、ブート手順を格納している記憶装置118、および、割り込みバス119より構成されている。

【0020】割り込み信号線110は、外部の入出力機器と割り込み制御装置112を接続している。外部機器が割り込みを発生すると、割り込み信号線110を経由して割り込み制御装置112が信号を受けとり、割り込み制御装置112は、この信号を数値化して、割り込みバス119を介してプロセッサ101に渡す。

【0021】クロック割り込み生成器111は、周期的な割り込みを生成する。

【0022】割り込み制御装置112は、外部機器からの割り込み要求を受け付け、要求元に仕掛けて数値化された割り込み信号を生成し、プロセッサ101に送る。また、プロセッサ101からの指示により、特定の

機器からの割り込み信号をプロセッサ101に通知しないようにできるとする。

【0023】プロセッサ101は、演算装置103、割り込みテーブルレジスタ104、ページテーブルレジスタ105、および、アドレス変換装置106より構成されている。

【0024】割り込みテーブルレジスタ104は、割り込みテーブル107の仮想アドレスを指し示している。割り込みテーブルの詳細については後述するが、割り込み番号毎の割り込みハンドラの開始アドレスを記録している。図1で、割り込みテーブルレジスタ104と割り込みテーブル107の接続を破線で記載しているのは、割り込みテーブルレジスタ104が割り込みテーブルの仮想アドレスを指し示すためである。割り込みが発生すると、プロセッサ101は割り込み制御装置112から数値化された割り込み番号を受ける。この番号をインデックスとして割り込みテーブル107より割り込みハンドラアドレスを取得し、割り込みハンドラに制御を渡す。

【0025】ページテーブルレジスタ105は、ページテーブル108を指し示している。ページテーブルレジスタ105は、ページテーブル108の物理アドレスを格納している。

【0026】アドレス変換装置106は、演算装置が要求する命令アドレス、あるいは、オペランドが格納されているアドレスを受けとり、ページテーブルレジスタ105の指しているページテーブル108の内容に基づき仮想-実アドレス変換を実施する。

【0027】計算機100には外部入出力装置としてキーボード113、ディスプレイ114、磁気ディスク115、その他の外部機器116、および、117が接続している。ディスプレイ114を除く機器は割り込み信号線110により割り込み制御装置112に接続している。

【0028】主記憶装置102の内容について簡単に説明する。現在、計算機101では2つのオペレーティングシステムが動作している。それぞれを第一のオペレーティングシステム、第二のオペレーティングシステムと呼ぶことにする。また、計算機を起動すると第一のOSが起動するように設定されており、外部機器116、および、117は第二のOSにより管理される機器であるとする。

【0029】第一のOSは、起動時の初期化処理で、その他のオペレーティングシステム用に、この場合は第二のOS用に物理メモリ領域を予約する。つまり、第一のOSが、第二のOS用に予約された物理メモリ領域を利用できないように物理メモリ領域を確保する。図1は、この奪った領域に第二のOSがロードされている様子を示している。

【0030】更に、第一のOSの初期化過程では、第一

のOSから外部機器116、および、117の利用する割り込み番号や入出力アドレスを、既に利用済みであるとして予約する。

【0031】また、第一のOSは、すべてのオペレーティングシステムから参照可能な共通領域を持つ。その共通領域に、割り込みテーブル107、割り込み管理プログラム、割り込みハンドラ、各オペレーティングシステムから呼びだし可能なインターフェイスモジュール等を格納する。

【0032】本発明の実施の形態の、動作概要について説明する。本実施の形態では、第二のOSは第一のOSよりも優先して動作する。優先して動作するとは、第一のOSは第二のOSがアイドル状態であるときのみ動作可能であることを示す。第二のOSの処理が終了しない限り、第一のOSは動作できない。

【0033】また、第二のOSが管理するデバイスが割り込みを発生すると、第一のOSの処理は中断され、制御は第二のOSに移る。第二のOS実行中に第一の割り込みが発生しても、その割り込み処理は第一のOSが実行されるまで延期される。

【0034】さらに、第一のOSと第二のOSは明確に区分されており、割り込みハンドラなどを配置する共通領域以外は、互いにアクセスできないようにする。これにより、2つのオペレーティングシステムが誤って互いの領域にアクセスして障害が起きることを防いでいる。

【0035】以下、上記の機能を実現する本発明の実施形態について説明する。

【0036】図2は、本発明の実施の形態における2つのオペレーティングシステムの間接的な関係を示した図である。それぞれのオペレーティングシステムは、それぞれ独立したアドレス空間を保持する。201は第一のOSの仮想空間で、202は第二のOSの仮想空間を示している。ここで、第二のOSの空間202に対応する実記憶は、図1の主記憶102の第二のOSの領域になる。

【0037】仮想空間の一部には、共通領域203がマップされる。共通領域203に対応する実記憶は、図1の主記憶102の共通領域として示した領域である。共通領域203は、もともとは第一のOSのカーネルの領域の一部である。第二のOSをロードする手順が、アドレス空間202を構築する時に、共通領域203をアドレス空間202にマッピングするように第二のOS用のページテーブルを作成する。この手順については後述する。

【0038】図2は、各オペレーティングシステムが管理するハードウェアを示している。第一のOSは、キーボード113、ディスプレイ114、および、磁気ディスク115を、第二のOSは入出力装置116、および、117を管理することを示している。クロック111と割り込み制御装置112は、もともとは第一のOS

が管理しているハードウェアであるが、共通領域203中のプログラムが管理することを示している。

【0039】次に、ページテーブルの構成について説明する。図3は、本発明の実施の形態でのページテーブルの構成を示している。

【0040】300がページテーブルである。ページテーブル300は、プロセッサ101の仮想アドレス空間の仮想ページ毎に、それぞれの仮想ページを記述するエントリを持っている。それぞれのエントリは、有効ビット301と、物理ページ番号302により構成される。

【0041】有効ビット301は、その仮想ページに対応する物理ページが割り当てられているか、つまり、仮想-実アドレス変換が可能かを示している。例えば、ページテーブル300の仮想ページ3は、有効ビットがセットされていないので、仮想ページ3に対応する物理ページが存在しないことを示している。有効ビット301がセットされていない仮想ページへのアクセスが発生すると、プロセッサはページフォルトを発生する。

【0042】物理ページ番号302は、仮想ページに対応する物理ページ番号を記録している。

【0043】アドレス変換装置106は、ページテーブルレジスタ105の指し示しているページテーブルの内容を参照して、演算装置103の生成する仮想アドレスを実アドレスに変換する。プロセッサ101は、変換により得られた実アドレスにより主記憶装置102を参照する。

【0044】ページテーブルを切替えることにより独立した空間を構築することができ、図2に示した第一のオペレーティングシステムの空間、および、第二のオペレーティングシステムの空間の構築が可能である。また、共通領域203については、両方のオペレーティングシステムのページテーブルの共通領域に対応する部分に、同じ物理ページをマップするように設定しておけば、共通領域を実現できる。

【0045】次に、割り込みテーブルの構成について説明する。図4は、割り込みテーブルの構成を示している。

【0046】400が割り込みテーブルである。割り込みテーブル400は、プロセッサ101が割り込み制御装置112から受ける割り込み番号毎の、割り込みハンドラの仮想アドレス401を記録している。プロセッサ101は割り込み要求を割り込み制御装置112から受けると、割り込み番号に対応する割り込みハンドラのアドレスを、割り込みテーブルレジスタ104の指し示している割り込みテーブル400から取得し、そのアドレスに制御を移すことで割り込み処理を開始する。

【0047】図5は、割り込み制御装置112を示している。割り込み制御装置112は、割り込みマスクレジスタ501、および、選択装置502を持っている。

【0048】割り込みを発生する入出力装置は、割り込

み信号線110により割り込み制御装置112と接続する。入出力機器の発生する割り込みは、割り込み信号線110のどの信号線に接続するかにより優先順位が付けられる。ここでは、割り込み0番に対応する割り込み信号がもっとも優先度が高い割り込みであるとする。

【0049】割り込み信号線110は、選択装置502に接続している。選択装置502は、割り込み信号を受けると、プロセッサがその割り込みを受け付けたことを通知するまで、未処理の割り込みがあることを記録している。

【0050】割り込みマスキレジスタ501は、入出力機器の発生した割り込みをプロセッサに通知してよいことを記録している。割り込みマスキレジスタ501は、プロセッサ101から入出力命令により設定可能である。

【0051】選択装置502は、割り込み信号線110から割り込み要求を受けた時と、割り込みマスキレジスタ501の内容が書き換えられた時に、選択装置502が記録している未処理割り込みと、割り込みマスキレジスタ502の内容を比較して、プロセッサに割り込みを通知するかどうかを決める。具体的には、選択装置502が記録している未処理割り込みのうち、割り込みマスキレジスタ501に割り込み可能と設定されていて、優先度の最も高い割り込みから順にプロセッサに通知する。選択した割り込みについて、選択装置502は、通知する割り込み信号に対応する数字信号を、割り込みバス119経由でプロセッサ101に送る。

【0052】プロセッサ101は、割り込みを受けた時に、入出力命令により選択装置502に記録されている未処理割り込み記録を解消できる。

【0053】次に、本発明の実施の形態における、計算機のブート手順について説明する。

【0054】ブート手順の始めの部分は、読みとり専用記憶装置である118に格納されている。記憶装置118は、プロセッサの物理アドレス空間のある決められたアドレスにマップされるようにバス109を介してプロセッサ101に接続している。この手順は、ハードウェア構成の検出、オペレーティングシステムカーネルをロードするプログラムの主記憶へのローディングを実施する。

【0055】プロセッサ101がリセットされると、プロセッサ101は予め定められた物理アドレスに制御を移す。記憶装置118は、この時に実行されるプログラムを格納しており、プロセッサ101がリセットされた時にこのプログラムに制御を渡せるように物理アドレス空間にマップされている。

【0056】記憶装置118に格納されているプログラムは、磁気ディスク装置112に格納されている第一のOSのカーネルローダを主記憶装置102にロードして実行する。カーネルローダは、磁気ディスク装置112の予め定められた位置にあり、記憶装置118に格納さ

れたプログラムは、容易にこれを見つけることができる。

【0057】カーネルローダの処理手順について説明する。図6は、本発明の実施の形態における、オペレーティングシステムのカーネルローダの処理手順を示すフローチャートである。

【0058】このカーネルローダは、オペレーティングシステムのファイルシステム構造を理解して、ファイル名よりファイルの格納位置を特定し、主記憶に読み込むことができるよう構成されている。

【0059】カーネルローダの処理手順について説明する。まず、カーネルにパラメータとして渡す主記憶リスト1101、ロードモジュールリスト1104、および、デバイスリスト1102を初期化し、カーネル用のページテーブル領域を割り当てる（ステップ601）。3つのリストの構成については後述する。

【0060】主記憶リスト1101は、主記憶装置102の利用状況を示すデータ構造であり、カーネルローダが以降の処理で物理メモリの割り当てをする場合は、主記憶リスト1101を参照、および、変更して実施する。

【0061】次に、ハードウェア構成の検査（ステップ602）、および、ハードウェア構成データの作成（ステップ603）を実施する。ステップ602においては、計算機100にどのような接続されているかハードウェアが検査する。続くステップ603では、ステップ602の結果に基づいてハードウェア構成に関するデータ構造であるデバイスリスト1102を作成する。オペレーティングシステムカーネルは、このデバイスリスト1102を参照してカーネル初期化処理を実施する。

【0062】次に、オペレーティングシステムカーネルの構成情報700を磁気ディスク装置112より読み込み、パラメータテーブル1100に構成情報のアドレスを設定する（ステップ604）。オペレーティングシステムのカーネルは、カーネル本体のファイルと、その他のデバイスドライバのファイルといったように、複数のファイルから構成されていても良い。構成情報700は、予め決められたファイル名で磁気ディスク112に格納されており、ロードプログラムはこれを見つけることができる。

【0063】本発明の実施形態におけるカーネル構成情報のデータ構造を図7に示す。700は、カーネル構成情報を記録しているファイルの内容を示している。構成情報ファイル700は、カーネルローダやオペレーティングシステムが参照するデータを格納している。格納されているデータには名前がつけられており、プログラムは名前からそれに対応するデータを取得することができる。図7の例では、名前がオブジェクトファイル（701）というエントリがあり、そのデータが702に格納されている。また、secondary OSには、第二のOS用の

データ(704)を格納しているとする。

【0064】カーネルローダの処理手順の説明を続ける。構成情報700を読み込んだ後、構成情報700中のオブジェクトファイルという名前のつけられたデータに格納されているカーネル構成ファイルのすべてについて、主記憶装置102に読み込み(ステップ606)、ロードモジュールリスト1104にエントリを追加し(ステップ607)、カーネル用のページテーブルの設定(ステップ608)を実施する。ここでは、kernel、driver1、および、driver2というファイル名のオブジェクトファイルをロードする。

【0065】ロードモジュールリストエントリの追加と、カーネル用のページテーブルの設定は、主記憶102にロードしたオブジェクトファイルに格納されているデータにしたがって実施する。カーネルを構成するオブジェクトファイルには、そのファイル内容をマップする仮想アドレス、ファイルの大きさなどが含まれている。これを参照してページテーブルを構築する。オブジェクトファイルのデータ構造については後述する。

【0066】最後に、ページテーブルレジスタ105を、構築したページテーブルのアドレスに設定し、プロセッサを仮想アドレス変換モードに移行させ(ステップ609)、構築した主記憶リスト1101、デバイスリスト1102、カーネル構成情報テーブル1103、および、ロードオブジェクトリスト1104の組から成るパラメータテーブル1110をパラメータとして、カーネルの初期化ルーチンに制御を渡す(ステップ610)。カーネルのエントリポイントは、カーネルファイル内のデータに記録されている。

【0067】次に、カーネルを構成するオブジェクトファイルの構造について説明する。図8は、本発明の実施の形態での、カーネルを構成するオブジェクトファイルの構造を示す図である。

【0068】800は、オブジェクトファイル全体を示している。オブジェクトファイル800は、801ないし811のヘッダ部分と、812ないし813のセクション部分より構成される。

【0069】ヘッダ部分の構成について説明する。ヘッダマップアドレス801とヘッダサイズ802は、オブジェクトファイル800ヘッダ部分のカーネル空間での格納位置を記述している。ヘッダ部分は、ヘッダマップアドレス801に記録されているアドレスに読み込まれる。

【0070】初期化エントリ803は、オブジェクトファイル800の初期化用ルーチンのアドレスを記録している。カーネルは、カーネル初期化時に各オブジェクトファイルの初期化ルーチンと呼ぶときに、各オブジェクトファイルの初期化エントリ803を参照して初期化ルーチンを見つける。

【0071】セクション数804は、オブジェクトファ

イル800に含まれているセクションの数を記録している。セクションとは、オブジェクトファイル内の連続しているデータ領域で、これを単位としてカーネルの仮想空間へのマッピングを決定する。例えば、オブジェクトファイルは、実行コードが格納されているセクションと、そのオブジェクトファイルが参照するデータを格納しているセクションを含んでいる。これらのセクションはオブジェクトファイル作成時にコンパイラにより作成される。

【0072】外部参照テーブルオフセット805と外部参照テーブルサイズ806は、このオブジェクトファイル内の実行コードが参照する、他のオブジェクトファイルの公開参照の情報を格納する外部参照テーブル810を記述している。外部参照テーブル810は、オブジェクトファイル800のヘッダ部分に含まれており、外部参照テーブルオフセット805は、ヘッダの先頭からの外部参照テーブル810のオフセットを記録している。

【0073】公開参照テーブルオフセット807と公開参照テーブルサイズ808は、このオブジェクトファイルが他のオブジェクトファイルの実行コードに公開しているモジュールとデータの情報を格納している公開参照テーブル811を記述している。公開参照テーブル811は、オブジェクトファイル800のヘッダ部分に含まれており、公開参照テーブルオフセット807は、ヘッダの先頭からの公開参照テーブル811のオフセットを記録している。

【0074】セクションデータ809は、オブジェクトファイル800に含まれる各セクションについてのデータを格納している。セクションデータは、セクション数804の数だけある。セクションデータの構成については後述する。

【0075】セクションデータの後に外部参照テーブル810と、公開参照テーブル811が続き、ヘッダ部分を構成する。

【0076】ヘッダ部分の後には、各セクションの本体812、813が格納されている。

【0077】セクションデータ809の構成について説明する。セクション開始オフセット820とセクションサイズ821は、オブジェクトファイル800内での、当該セクションの開始オフセットと、当該セクションの大きさを記録している。

【0078】セクションは、セクションマップアドレス822に記録されたアドレスに配置されるようにカーネルの仮想空間にマップされる。セクション名称823には、当該セクションの名前を示す文字列が格納されている。

【0079】外部参照テーブルの構造について説明する。図9は、外部参照テーブルの構造を示している。テーブル810の先頭には、このテーブルに含まれる外部参照情報の数901が格納されている。

【0080】続いて、オブジェクトファイル名902、外部参照名903が格納されている。オブジェクトファイル名902と外部参照名903は、文字列テーブル905へのオフセット値を格納しており、実際の文字列による名称は文字列テーブル905内に格納されている。

【0081】外部参照アドレス904には、この当該外部参照エントリで記述される外部参照の実際のアドレスが格納される。カーネルは、オブジェクトファイルを主記憶にロードするときに、当該の外部参照を含むオブジェクトファイルの公開参照のテーブルを参照して関数、あるいは、データのアドレスを取得し、外部参照アドレス904に設定する。オブジェクトファイルの実行コードは、外部関数アドレス904に格納されたアドレスを参照して、他のオブジェクトファイル内の関数の呼び出しや、データの参照をするようコンパイルされており、他オブジェクトモジュールにある関数の実行や、データ参照が可能である。

【0082】オブジェクトファイル名902、外部参照名903、および、外部参照アドレス904が1つの外部参照を定義し、外部参照数901に記録されている数だけ、これらのエントリが連続して配置される。その後、文字列テーブル905が格納される。文字列テーブルは、オブジェクトファイル名や、外部参照名の文字列を格納している。

【0083】公開参照テーブルの構造について説明する。図10は、公開参照テーブルの構造を示す図である。

【0084】テーブル811の先頭には、この公開参照テーブル811により他のオブジェクトモジュールに公開される参照名の数1001が記録されている。1つの公開参照は、公開参照名1002と公開参照アドレス1003により記述される。公開参照名1002は、文字列テーブル1004へのオフセット値を格納しており、実際の文字列による名前は文字列テーブル1004に格納されている。公開参照アドレス1003は、この参照に対応するアドレスを格納している。

【0085】次に、ステップ601から始まるブート手順が作成するハードウェア構成データと、ロードオブジェクトデータの構成について説明する。図11がハードウェア構成データとロードオブジェクトデータの構成を示す図である。

【0086】パラメータテーブル1100は、カーネルローダが作成するデータ構造である。パラメータテーブル1100から始まる3つのリストは、ローダが構築するカーネルの仮想空間に配置されるので、カーネルから参照可能である。

【0087】パラメータテーブル1100は、ローダが構築した3つのリストの先頭へのポインタと、1つのテーブルへのポインタを保持している。3つのリストとは、主記憶リスト1101、デバイスリスト1102、

および、ロードオブジェクトリスト1104で、1つのテーブルはカーネル構成情報テーブル1103である。それぞれについて説明する。

【0088】主記憶リスト1101は、主記憶ブロック記述データ1110のリストである。主記憶ブロック記述データ1110は、ベースアドレス1111、ブロックサイズ1112、ブロック利用状況1113、および、次の主記憶ブロック記述データへのポインタ1114から構成されている。

【0089】主記憶ブロック記述データは、ある連続した主記憶領域についての利用状況を記録している。ベースアドレス1111は連続領域の開始物理アドレスを示し、ブロックサイズ1112は連続領域の大きさを格納している。ブロック利用状況1113は、当該連続領域が未使用であるか、あるいは、ローダにより割り当て済みであることを示す値が格納されている。そして、次エントリへのポインタ1114によりリストを構成している。図11では、1110の次のエントリは1120である。主記憶リスト1101を参照することで、物理メモリの利用状態を知ることができる。

【0090】デバイスリスト1102は、カーネルローダが検出したハードウェアデバイスに関するデータを格納しており、ステップ603で作成されている。デバイスリスト1103は、デバイスデータ1150からなるリストである。デバイスデータ1150は、デバイスタイプ1151、デバイス情報1152、および、次のデバイスデータへのポインタ1153より構成される。

【0091】デバイスタイプ1151は、このデバイスデータエントリ1150により記述されるデバイスの種類を示す値が格納されている。デバイス情報1152は、デバイスの種類に特有なデータを格納している。例えば、割り込み番号やI/Oアドレスなどがそれに相当する。そして、次エントリへのポインタ1153によりリストを構成している。

【0092】カーネル構成情報テーブルへのポインタ1103は、カーネルローダが主記憶102に読み込んだカーネル構成情報ファイル700の内容を指し示している。

【0093】ロードオブジェクトリスト1104は、カーネルローダが主記憶にロードしたオブジェクトファイルに関するデータを保持している。ロードオブジェクトリストは、ロードオブジェクトデータ1130のリストである。ロードオブジェクトデータ1130は、オブジェクトファイル名1131、オブジェクトアドレス1132、および、次のロードオブジェクトデータへのポインタ1133より構成されている。

【0094】オブジェクトファイル名1131は、ロードオブジェクトデータ1130により記述されているオブジェクトファイルのファイル名である。オブジェクトアドレス1132は、当該オブジェクトファイルのヘッ

領域がロードされているカーネル空間のアドレスを格納している。そして、次エントリへのポインタ 1133 によりリストを構成している。

【0095】ロードオブジェクトリスト 1104 は、カーネルローダがカーネルを構成するオブジェクトファイルを読み込む時に同時に作成している（ステップ 607）。

【0096】次に、本発明の実施の形態における第一の OS の初期化手順について説明する。図 12 は、第一の OS の初期化手順を示すフローチャートである。

【0097】まず、パラメータとして渡されたパラメータテーブル 1100 中のロードオブジェクトリスト 1104 を参照して、カーネルローダがロードしたオブジェクトファイルの外部参照アドレス解決を実施する（ステップ 1201）。アドレス解決では、各オブジェクトファイルにある外部参照テーブル 810 の、外部参照アドレス 904 を決定する。アドレスは、各オブジェクトファイルの公開参照テーブル 811 を参照して決定する。

【0098】続く、ステップ 1202 では、カーネル起動時のパラメータとして渡されたパラメータテーブル 1100 の主記憶リスト 1101 を参照して、第二の OS 用に主記憶領域を確保する。

【0099】具体的には、カーネル構成情報テーブル 700 より第二の OS の情報を取り出す。図 7 の例では、第二の OS の構成情報は 704 に格納されている。この構成情報 704 を参照して、確保すべき主記憶の大きさを決定する。そして、主記憶リスト 1101 の空きブロックエントリの内容を変更して、主記憶領域を割り当てる。この処理は、第一の OS が空きメモリ管理を始める前に実施する。

【0100】これにより、第一の OS から見ると、第二の OS に割り当てた主記憶領域は存在しないことになり、第一の OS から参照されることがなくなる。したがって、割り当てた領域は、第二の OS が自由に使うことのできる主記憶領域となる。これは、図 1 の第二の OS の領域に相当する。

【0101】次のステップ 1203 では、カーネル内部のデータ構造の初期化を実施する。この初期化には、後で述べるデバイス管理テーブルの初期化も含む。

【0102】ステップ 1204 では、第二の OS が管理するデバイスを予約する。ここで予約するとは、第一の OS から利用できないようにすることである。具体的には、第一の OS が管理しているデバイス管理テーブルへの登録を実施する。

【0103】第二の OS が管理するデバイス資源は、パラメータテーブル 1100 のカーネル構成情報テーブル 1103 の指すテーブル 700 に格納されている、第二の OS の構成情報を参照して決める。この実施形態では、図 7 の 704 に格納されているデータがそれに相当する。

【0104】デバイス管理テーブルについて説明する。図 13 は、第一の OS のデバイス管理テーブルの構造を示した図である。デバイス管理テーブルは、割り込みベクタ管理テーブル 1300 と、I/O アドレス管理リスト 1310 の 2 つのデータ構造からなる。

【0105】割り込みベクタ管理テーブル 1300 は、プロセッサ 101 が受け付ける各割り込み番号について、第一の OS がその割り込み番号を利用するかどうかを示す値を格納している。カーネルは、デバイスドライバが初期化時に割り込み番号を要求した場合に、このテーブル 1300 を検査し、要求された割り込み番号が利用されているか検査し、そうでない場合にのみ要求された割り込み番号を使用する権利をデバイスドライバに与える。既に利用済みであると記されている場合は、そのデバイスは第一の OS からは利用できないことになる。

【0106】図 2 の入出力装置 116 と 117 を例として説明する。入出力装置 116 と 117 は、それぞれ割り込み番号 4 と 5 と要求すると仮定する。入出力装置 116 と 117 は、第二の OS が管理するデバイスである。入出力装置 116 と 117 の要求する割り込み番号は、カーネル構成情報テーブル 700 の第二の OS の構成情報 704 に記録されている。ステップ 1204 では、この構成情報 704 を参照して、割り込みベクタ管理テーブルの、割り込み番号 4 と 5 のエントリに利用中であることを示す値を格納する。この処理は、第一の OS がデバイス管理を開始する前に実施するため、第一の OS は、入出力装置 116 と 117 にアクセスすることができなくなり、装置 116 と 117 を第二の OS の管理下におくことができる。

【0107】I/O アドレス管理リスト 1310 についても同様である。I/O アドレス管理リスト 1310 は、I/O アドレス範囲を表現するエントリ 1320 からなるリストである。エントリ 1320 は、第一の OS が利用する I/O アドレス範囲 1321 と、リストを構成するための次のエントリへのポインタ 1322 からなる。割り込みベクタ管理テーブル 1300 と同様、デバイスドライバが初期化時に I/O アドレス範囲を要求した場合、カーネルは、そのアドレス範囲が既に利用されているか I/O アドレス管理リスト 1310 により検査し、未使用である場合、このリスト 1310 にエントリを追加して、利用許可を与える。

【0108】第二の OS が管理するデバイスが要求する I/O アドレス範囲は、割り込み番号と同様にカーネル構成情報テーブル 700 に格納されているので、それを参照すれば要求アドレスを知ることができ、第一の OS がデバイス管理を開始する前に I/O アドレスを予約できる。

【0109】ステップ 1202 の処理により、第一の OS から完全に独立した第二の OS 専用の空間を構築することが可能になる。さらに、ステップ 1204 の処理に

より、第一のOS上で動作するユーザプログラムは、第二のオペレーティングシステムが管理するデバイス、この例では、入出力装置116と117へのアクセスが不可能になる。また、装置116と117の割り込み番号とI/Oアドレスを利用するデバイスドライバを導入することを禁止できる。

【0110】この2つのステップの処理の効果として、第一のOSが関知しない部分に第二のOSを導入することが可能になる。

【0111】続くステップ1205、ないし、ステップ1207は通常のオペレーティングシステムの初期化処理と同じである。ステップ1205のシステムデバイスの初期化では、カーネルが直接管理するシステムデバイスの初期化を実施する。システムデバイスとは、クロック割り込みなど、第一のOSの実行に不可欠で、第一のOSが必ず存在していると仮定しているデバイスである。

【0112】ステップ1206では、カーネルローダがロードしたオブジェクトファイルについて、それぞれの初期化エントリを実行する。初期化エントリアドレスは、オブジェクトファイルのヘッダ部分に格納されている。最後に、初期プロセスを作成する(ステップ1207)。

【0113】次に、本発明の実施の形態における、第二のOSのロード手順について説明する。図14は、第二のOSのロード手順を示すフローチャートである。

【0114】まず、第二のOS用に割り当てた物理メモリ領域に、第二のOSのオブジェクトファイルを読み込む必要がある。しかし、第二のOSの物理メモリ領域は、そのままでは第一のOSから書き込むことはできないので、割り当てた物理メモリ領域を第一のOSの仮想空間に一時的にマッピングする(ステップ1401)。

【0115】ステップ1402では、マッピングした領域に、第一のOSのファイル読み込み手順を利用して、第二のOSのオブジェクトファイルを読み込む。なお、第二のOSのオブジェクトファイルの形式は、第一のOSのオブジェクトファイル形式800と同じ形式であるとする。

【0116】次に、第二のOS用のページテーブルを作成する(ステップ1403)。このページテーブルも第二のOS用の領域に作成する。この時に、第一のOSと共有する部分について、第二のOSの空間からも参照できるように、ページテーブルを構築する。

【0117】共通領域203について、割り込み処理や共通データの管理を実施するデバイスドライバ(以下サポートドライバ)をロードした領域を共通領域203とする。このデバイスドライバがロードされたアドレスは、ロードオブジェクトリスト1104より知ることができる。また、続くステップ1404で、第二のOSのカーネルの外部参照を解決する。但し、第二のOSが直

接参照できる他のオブジェクトファイルの参照は、共通領域203に配置されている関数とデータ、つまり、サポートドライバの公開参照のみである。したがって、ここでは、サポートドライバのオブジェクトファイルのヘッダ部分に格納されている公開参照テーブル811を参照して、第二のOSのカーネルオブジェクトファイルの外部参照テーブル810の外部アドレス904を決定する。

【0118】次に、第二のOSの公開参照のアドレスを、共通領域のデータ領域に割り当てられた外部参照アドレステーブルに書き込む。共通領域となっているサポートドライバは、第一のOSのデバイスドライバとして、第一のOSの機構にしたがって読み込むため、第二のOSの公開参照とリンクすることはできない。

【0119】ここでは、サポートドライバのデータ領域内に必要な外部参照名と、それに対応する外部アドレスを格納するテーブルを予め用意しておく。サポートドライバの実行コードは、このテーブルを参照して第二のOSのカーネルの公開関数の呼びだし、公開データの参照を実施するように記述する。そして、第二のOSのロード時に、このテーブルの外部アドレス欄にサポートドライバの公開参照のアドレスを書き込むこととする。

【0120】これで、第二のOS領域の設定を終了し、第一のOSのカーネル領域にマップした、第二のOS用の物理メモリ領域のマッピングを解除する(ステップ1406)。

【0121】次に、OSコンテキストテーブル1510の第二のOSのコンテキストと、OS識別変数1530を設定する(ステップ1407)。OSコンテキストは、実行オペレーティングシステムを切り替えるときに参照するデータ構造で、ページテーブルアドレス値とスタックポインタの初期値とで構成する。ここでは、ページテーブルレジスタ値として第二のOSをマップするページテーブルのアドレスを、スタックポインタ値として第二のOSのカーネルスタックの初期アドレスを設定する。OS識別変数1530には、第一のOSが実行中であることを示す値を格納する。OSコンテキストテーブル1510とOS識別変数1530については後述する。

【0122】次に、第二のOSの初期化モジュールを実行する(ステップ1408)。これには、オペレーティングシステム空間の切替が伴う。オペレーティングシステムの切替えについては、別のフローチャートにより説明する。また、第二のOSの初期化モジュールは公開参照になっており、サポートドライバはそのアドレスを知ることができる。

【0123】最後に、ステップ1409にて、現在の割り込みテーブル104に登録されている第一のOSの割り込みハンドラのアドレスを、割り込み識別テーブル1520のハンドラの欄1522にコピーし、割り込みテ

ーブルレジスタ値を、サポートドライバに割り当てた割り込みテーブルのアドレスに変更する。これは、プロセッサ 101 の割り込みテーブルレジスタ 104 の変更により実施する。

【0124】割り込みテーブルをサポートドライバ内のテーブルに変更するのは、割り込み発生時にどちらのオペレーティングシステムが実行していても、常にプロセッサ 101 の仮想アドレス空間に割り込みテーブルが存在している必要があるためである。割り込みテーブルに登録される割り込みハンドラも、サポートドライバ内に配置する。サポートドライバの領域は、ステップ 1403 にて、第二の OS の仮想空間にもマッピングして共通領域 203 とするので、いつでも参照できることになる。サポートドライバの割り込み処理については後述する。

【0125】また、ステップ 1409 では、第一の OS の割り込み管理情報も変更する。具体的には、割り込み禁止レベルに関連するデータ構造を変更するが、これについては後述する。

【0126】共通領域 203 に格納するデータ構造について説明する。図 15 は、共通領域 203 のうちのデータ領域 1500 に格納するデータ構造を示した図である。図 15 にしたがって順に説明する。

【0127】1510 は、OS コンテキストテーブルである。OS コンテキストテーブル 1510 は、第一の OS と第二の OS との間の切替えに必要なデータを保持する。この実施の形態では、第一の OS は第二の OS がアイドル状態の時のみ走行できるとする。この場合、第一の OS 実行中のある時点で第二の OS への切替が起こり、第二の OS の実行が終了した時点で、第一の OS に制御を戻せば良い。

【0128】したがって、それぞれで保存しておかなければならないコンテキストは 1 組で良い。第一の OS のコンテキストについては、OS 切替が要求された時点でのページテーブルレジスタ値 1511 と、スタックポインタ値 1512 を保存しておけば、第二の OS 実行終了後に、第一の OS に制御を復帰させることができる。

【0129】また、第一の OS から第二の OS へ制御を切り替えるときには第二の OS は動作していない。したがって、第二の OS のコンテキストは、ページテーブルアドレスもスタックポインタも固定の値でよい。第二の OS のページテーブルレジスタ値 1513 とスタックポインタ値 1514 は、第二の OS をロードするときに設定する（ステップ 1407）。

【0130】1520 は、割り込み識別テーブルである。割り込み識別テーブル 1520 は、外部割り込みの割り込み番号毎に、どちらのオペレーティングシステムが割り込みを処理するかを示す値 1521 と、割り込みハンドラのアドレス 1522 が記録されている。外部割り込みが発生すると、共通領域 203 内の割り込みハンド

ラが割り込みを捕獲し、この割り込み識別テーブル 1520 の処理 OS 1521 を参照して、どちらの OS に処理させるかを決定し、ハンドラ 1522 のアドレスへ制御を渡す。割り込み処理の詳細については、後述する。

【0131】1530 は、実行中のオペレーティングシステムを示す値を格納している OS 識別変数である。この変数 1530 は、ステップ 1601 から始まる OS 切り替え手順で OS 切り替えの度に設定する。割り込み処理では、この変数 1530 を参照して割り込み処理手順を決定する。

【0132】1540 は、第二の OS の実行中に、第一の OS が管理しているデバイスの割り込みが発生したかを示す遅延割り込み状態変数である。この変数 1540 は、どの割り込み番号の割り込みが発生したかを記録している。OS 切り替え手順は、第二の OS の実行が終了したときにこの変数 1540 を検査して、割り込み処理を起動するか決定する（ステップ 1608）。

【0133】オペレーティングシステムの切替え手順について説明する。図 16 は、本発明の実施の形態における、オペレーティングシステムの切替え手順を示すフローチャートである。この切り替え手順は、第一の OS の実行中に呼び出され、第二の OS の切り替えを実施する。

【0134】図 16 に示した手順は、第二の OS への切り替え後に実行する第二の OS のモジュールのアドレスと、そのモジュールへ渡す引数を引数として受ける。第二の OS のモジュールのアドレスは、共通領域 203 内に設定した、外部参照アドレステーブルを参照すれば知ることができる。

【0135】まず、始めのステップ 1601 で、現在のスタックポインタ値とページテーブルレジスタ値を、OS コンテキストテーブル 1510 の、第一の OS のコンテキストとして保存する。ステップ 1601 では、現在のスタックポインタ値を 1512 に、現在のページテーブルレジスタ 105 の値を 1511 に保存する。

【0136】他のレジスタコンテキストについて、OS コンテキストテーブル 1510 に保存する必要はない。必要があれば、第一の OS のスタックに保存すればよい。

【0137】スタックポインタとページテーブルレジスタ値を保存した後、ステップ 1602 にて、ページテーブルレジスタ 105 に第二の OS を仮想空間にマップするページテーブルのアドレスを設定する。これは、OS コンテキストテーブル 1510 の 1513 に記録されている。更に、スタックポインタを第二の OS 用に設定する。これも、テーブル 1600 の第二の OS のスタックポインタ 1514 に格納されている。

【0138】次のステップ 1603 で、第一の OS の割り込み状態を示す遅延割り込み状態変数 1540 をクリアする。状態変数 1540 は、第二の OS 実行中に発生

した、第一のOSが管理しているデバイスからの割り込みの発生状況を記録する変数である。第二のOSを実行する前に、これをクリアしておく。

【0139】そして、現在実行中のOSの示しているOS識別変数1530を、第二のOSを示す値に書き換える(ステップ1604)。スタックポインタ、ページテーブルレジスタ105、および、OS識別変数1530は、常に一貫した値になっていなければならないので、ここまでのステップ1601、ないし、1604は全ての外部割り込みを禁止した状態で実行しなければならない。

【0140】続くステップ1605で、引数として渡されたモジュールのアドレスへ制御を移し、第二のオペレーティングシステムに制御を渡す。本発明のこの実施の形態においては、第一のOSは第二のOSが実行していない時、つまり、第二のOSがアイドル状態のときだけ実行できるとする。したがって、第二のOSの処理が終了した時に、ステップ1606へ制御が戻る。

【0141】ステップ1606では、ステップ1601でOSコンテキストテーブル15100に保存したページテーブルレジスタ値1511と、スタックポインタ値1512のそれぞれを回復する。続くステップ1607で、OS識別変数1530を第一のOSが実行中であることを示す値に変更する。この2つのステップの処理も割り込みを禁止した状態で実行しなければならない。

【0142】次に、第二のOSの実行中に発生した、第一のOSが管理するデバイスの外部割り込みを処理する。まず、ステップ1608では、遅延割り込み状態変数1540を検査して、割り込みが発生したかどうかを検査する。発生していない場合は、OS切替え手順は終了し、呼出元に復帰する。

【0143】そうでない場合、割り込みが発生している場合は、ステップ1609を実行する。このステップでは、第二のOSの実行中に発生した割り込みを、第一のOSが管理している延期割り込み状態変数に、未処理の割り込みがある旨を記録する。続いて、第一のOSの割り込み処理を起動する(ステップ1610)。全ての割り込み処理が終了した時に、OS切替え手順の呼出元に復帰する。

【0144】本発明の実施の形態における割り込み処理について説明する。図17は、本実施形態の割り込み処理手順を示すフローチャートである。この手順を実行するモジュールは、割り込みハンドラとしてプロセッサの割り込みテーブル107に登録される。さらに、この割り込みハンドラは、両方のオペレーティングシステムから参照できる共通領域203に配置する。

【0145】外部割り込みが発生して、プロセッサ101により割り込みハンドラが起動されると、割り込みハンドラは割り込み要因を検査し、割り込みを発生したデバイスが第一のOSが管理するデバイスか、第二のOS

が管理するデバイスか判定する(ステップ1701)。この判定は、割り込み識別テーブル1520を割り込み番号をインデックスとしてOS欄1521を参照することより実施する。第一のOSのデバイスである場合はステップ1702へ、第二のOSのデバイスの場合はステップ1705へ進む。例えば、図15でいえば、割り込み番号が1であれば第一のOSの割り込みであり、割り込み番号4であれば第二のOSの割り込みとなる。

【0146】割り込みが第一のOSのデバイスの割り込みである場合、ステップ1702を実行する。ステップ1702では、割り込み発生時に実行していたOSを判定する。この判定は、OS識別変数1530を参照して実施する。実行中のOSが第一のOSの場合はステップ1703へ、第二のOSの場合はステップ1704へ進む。

【0147】ステップ1703から始まる処理は、第一のOSが管理しているデバイスが、第一のOSを実行中に割り込みを発生した場合の処理である。ステップ1703では、あたかもステップ1701から始まる処理が存在せず、第一のOSの割り込みハンドラが、直接プロセッサ101から制御を受けたように見えるようにコンテキストを設定する。ここでコンテキストとは、スタックの内容やレジスタの内容を示す。そして、第一のOSの割り込みハンドラへ制御を渡す。第一のOSの割り込みハンドラのアドレスは、割り込み識別テーブル1520のハンドラ欄1522に格納されている。例えば、割り込み番号1の割り込みであるならば、1をインデックスとして割り込み識別テーブルを参照して、ハンドラアドレスを求める。

【0148】この場合、ステップ1701から始まる手順には制御は戻らず、第一のOSが処理を続ける。

【0149】第一のOSが管理しているデバイスが、第二のOSを実行中に割り込みを発生した場合、ステップ1704を実行する。ステップ1704では、遅延割り込み状態変数1540に割り込みを発生したデバイスの割り込み番号を記録する。割り込みハンドラの処理はこれで終了する。この場合の割り込みの処理は、実行OSが第一のOSに切り替わったときに実行される(ステップ1608)。

【0150】発生した外部割り込みが、第二のOSが管理するデバイスの割り込みだった場合、ステップ1705へ進み、どちらのOSが実行中であるか検査する。ここでも、OS識別変数1530によって実行中のOSを判定する。第一のOSが実行中の場合は、ステップ1706へ、第二のOSが実行中の場合はステップ1711へ進む。

【0151】第二のOSが管理するデバイスの割り込みが、第二のOSの実行中に発生した場合、ステップ1711を実行する。ステップ1711は、第二のOSの割り込みハンドラを起動する。第二のOSの割り込みハン

ドラのアドレスは、割り込み識別テーブル1520のハンドラ欄1522に記録されている。第二のOSの割り込みハンドラ処理が終了して制御が戻ってきたら、この割り込みハンドラも終了し、割り込まれた時のコンテキストを回復して制御を元に戻す。

【0152】第二のOSが管理するデバイスの外部割り込みが、第一のOSの実行中に発生した場合、ステップ1706を実行する。この場合は、第一のOSの実行よりも第二のOSの処理を優先して実行する。

【0153】まず、ステップ1706では、コンテキストを保存する。ここでのコンテキストとは、割り込み処理が終了した後で第一のOSに戻すときに、割り込まれたときの状態を回復するのに必要なスタックの内容とレジスタの内容を示す。このコンテキストは、第一のOSのカーネルのスタックに保存する。

【0154】続いて、実行OSの切り替えと第二のOSの割り込み処理の起動を実行する（ステップ1707、1708）。これは、ステップ1601から始まる手順により実行する。

【0155】第二のOSの処理が終了した時点で、第一のOSへの切り替えを実行し（ステップ1709）、割り込み時のコンテキストを回復し（ステップ1710）、第一のOSの処理を再開する。ステップ1709の処理は、必ずしも、ステップ1701から始まる処理と同一のモジュール内で実行されなくてもよい。第一のOSへの切り替えにより処理はこのモジュールへ復帰する。

【0156】2つのオペレーティングシステムで共有しているクロック割り込みの処理について説明する。クロック割り込みは、共通領域内の割り込みハンドラにより捕獲する。この割り込みハンドラでは、まず、第二のOSのクロック割り込み用の割り込みハンドラを実行する。第二のOSの割り込みハンドラはハンドラ2欄1523に格納されている。第二のOSの割り込みハンドラの実行が終了したら、図17のステップ1702から始まる処理により第一のOSの割り込み処理を実行する。第一の割り込みハンドラのアドレスはハンドラ欄1522に格納されている。

【0157】次に、第一のOSの割り込み制御部分について説明する。これは、第一のOSの割り込み制御によって、誤って第二のOSが管理するデバイスの割り込みが禁止にされてしまわないようにするための処理である。

【0158】第一のOSは、割り込み禁止レベルにより割り込みを制御しているとする。割り込み禁止レベルは、オペレーティングシステムのカーネル内の割り込み処理の延長で動作する部分と、そうでない部分との間の排他制御を実現するために必要となる機構である。

【0159】第一のOSは、割り込み制御装置112をプログラムすることで割り込み禁止レベルを実現する。

つまり、割り込み制御装置112の割り込みマスクレジスタ502をプログラムして、選択的に外部割り込みをマスクにする。第一のOSは第二のOSについて全く知らないで、第一のOSが割り込み禁止レベルを変更したときに、第二のOSのデバイスの割り込みがマスクされてしまう可能性がある。これを防ぐために、第一のOSの割り込み制御部分を変更する。

【0160】図18は、割り込み禁止レベルを実現する第一のOSが管理しているデータ構造を示している。1800は、割り込み禁止レベルテーブルである。それぞれの割り込みレベルは数値で表現され、それぞれの割り込み禁止レベルについて、何番の外部割り込みをマスクするかを示している。テーブル1800のチェックのつけられているところは割り込みをマスクする設定することを示している。例えば、割り込み禁止テーブル1800では、割り込み禁止レベル0では、どの割り込みもマスクされないことを示している。また、割り込み禁止レベル3では、割り込み番号3、ないし、5の割り込みを割り込み制御装置112によりマスクすることを示している。割り込み禁止レベル5では、すべての割り込みが割り込み制御装置112によりマスクされる。

【0161】本発明では、第二のOSの初期化時に、この割り込み禁止レベルテーブル1800を変更する（ステップ1409）。ステップ1409では、第二のOSが管理するデバイスが発生する割り込みについて、第一のOSがそれらの割り込みをマスクしないように割り込み禁止レベルテーブルを変更する。具体的には、割り込み識別テーブル1520のOS欄1521を参照して、第二のOSが管理する割り込み番号について、割り込み禁止レベルテーブル1800のチェックをクリアする。

【0162】この例では、割り込み番号4と5が第二のOSが処理する割り込みとなっている。したがって、割り込み禁止レベルテーブル1800のすべての割り込み禁止レベルの、割り込み番号4と5の欄（1801と1802のすべて）をクリアする。

【0163】これにより、第一のOSが割り込み禁止レベルを変更しても、第二のOSが管理するデバイスの割り込みはマスクされなくなる。

【0164】以上により、一台の計算機で2つのオペレーティングシステムを同時に動作させることが可能になる。

【0165】本発明によれば、第一のOSに変更を加えて2つのオペレーティングシステムを同時に動作させる場合、変更箇所がオペレーティングシステムカーネルの初期化部分、デバイス資源予約、および、割り込み禁止制御部分に限定されるため、簡単に2つのオペレーティングシステムを動作させることが可能になる。

【0166】仮想計算機による方式では、物理メモリやI/Oチャネルを仮想化するために特権命令のエミュレーションが必要になるが、これをソフトウェアで実現す

るとオーバーヘッドが大きくなり問題である。仮想計算機方式では、このオーバーヘッド削減のために、特殊なハードウェアを持つことが多くなっている。しかし、本発明では、それぞれのデバイスについて、デバイスを管理するオペレーティングシステムを予め決定し、さらに、物理メモリについて利用できる範囲を初期化時に決定することにより、オペレーティングシステム同士が互いに干渉しないようにして、仮想計算機でのような複雑なソフトウェアによる制御を廃止し、高速化のためのハードウェアも不要とした。

【0167】本発明によれば、第一のOSの機能を補完するOSを容易に追加することが可能である。従来技術においても、第一のOSの構成要素として、例えばデバイスドライバとして新たな機能をカーネルに追加することは可能である。しかし、第一のOSの構成要素としてしまったのでは、その構成要素は第一のOSの管理下でしか動作できない問題がある。つまり、第一のOSが障害により停止してしまったとき、追加した機能モジュールも動作することはできないのである。

【0168】本発明によれば、新たな機能を実現する構成要素を、第一のOSと独立して構成することができ、第一のOSが停止してしまったとしても、その機能モジュールだけは継続して動作することが可能になる。この実施形態については、後述する。信頼性を要求される機能モジュールを第二のOSとして組み込めば、第一のOSが停止してしまったときでも、何らかの回復処理を実現させるといったことが可能になる。このように本発明は、計算機システムの高信頼化を実現する手段となり得る発明である。

【0169】また、この実施形態では、第二のOSが第一のOSに優先して処理を実行すると説明した。第一のOSは、第二のOSがアイドル時にしか動作できないことや、第二のOSの割り込みはいつでもすぐに処理することにより、第二のOSを優先させている。これにより、第一のOSが実時間処理に適合していなくても、第二のOSとして実時間処理に向くOSを導入すれば、第一のOSの特徴を活かしたままで、実時間処理性能に優れた計算機システムを構築することが可能になる。例えば、第一のOSが優れたGUI(Graphical User Interface)を持っているが実時間処理性能に欠ける場合、第一のOSよりも優先して動作する実時間処理向きオペレーティングシステムを第二のOSとして導入してやることで、GUIにも優れ、実時間処理にも優れた計算機システムの構築ができる。

【0170】このように、本発明は、特別なハードウェアの支援なしで、第一のOSに欠けている機能を容易に導入する方法であり、さらに、その機能は第一のOSとは全く独立して動作させることを可能にする。

【0171】次に、本発明の第二の実施形態について説明する。第二の実施形態は、これまで説明してきた実施

の形態の拡張である。この実施の形態では、第一のOSが障害により停止しても動作しつづける第二のOSの導入が実現可能である。

【0172】第一の実施形態に加えて、第一OS実行状態変数1550を共通領域に置く。この変数1550は、第一のOSが通常動作しているか、そうでないかを示す値を格納している。この変数1550は、第二のOSをロードする時の処理で、通常動作を示す値に初期化する。

【0173】図19は、本発明の第二の実施の形態の第一のOSの停止処理手順を示すフローチャートである。この処理手順は、第一のOSの停止処理を実行するモジュールを変更して実装する。

【0174】まず、第一のOSの停止処理モジュールに制御がきたら、第一OS実行状態変数1550を第一のOSが停止していることを示す値に設定する(ステップ1901)。その後、第一のOSの停止処理を実行する(ステップ1902)。最後に、第一のOSへの割り込みをマスクして、第二のOSが処理するデバイスの割り込みを許可して(ステップ1903)、割り込みが発生するまで待つ(ステップ1904)。割り込みが発生すると、実行OSの切り替えられ、第二のOSが処理を実行する。

【0175】更に、実行OS切り替え手順を変更する。第一の実施の形態では、ステップ1601から始まる手順により実行OSの切り替えを実行した。第二の実施の形態では、この手順で第二のOSのモジュールを実行した後、つまり、ステップ1605の後で、第一OS実行状態変数1550を検査する。ここで、第一OS実行状態変数1550が、第一のOSが停止していることを示す値になっているならば、ステップ1606以降の処理を実行せずに、割り込み待ちを実行する。

【0176】以上のデータ構造と手順により、第一のOSが停止しても第二のOSの実行を継続することが可能になる。この実施形態では、第一のOSの停止処理モジュールを変更するとしたが、第一のOSがエラーにより停止したときの停止処理過程で実行されるモジュールを変更して、第一のOSの停止を検出して割り込み待ちをしても同様の効果を実現できる。

【0177】本発明の第3の実施の形態について説明する。これまで説明してきた実施の形態では、カーネル本体を変更することにより2つのOSの同時実行等を実現してきた。第3の実施の形態では、カーネル本体を変更せずに、前記の実施の形態の機能を実現する。

【0178】様々な種類のハードウェアをサポートするオペレーティングシステムでは、ハードウェア依存の処理がカーネル本体からは切り離されて、別のオブジェクトファイルとして構成されている場合がある。例えば、割り込み制御装置112が計算機により異なる場合や、バス109の構成が異なってI/Oアドレス空間が計算

機により異なる場合である。

【0179】図20は、このようなオペレーティングシステム、つまり、割り込み制御装置やバスなどの基盤となるハードウェアの違いを吸収するためのコードやデータがカーネル本体とは分離されたオブジェクトファイルにある場合の、カーネル領域の様子を示した図である。

【0180】カーネル領域2000には、プロセッサ101のカーネルモードで実行されるモジュールや、オペレーティングシステムが管理するデータ構造がある。カーネル本体2001は、メモリ管理、プロセススケジューリング、および、ファイルシステムなどの、ハードウェア非依存の処理を実施するコードやデータを持っている。カーネル本体2001とハードウェア依存部2002の間には、ハードウェア依存部2002の提供しなければならないモジュールと、カーネル本体2001が提供するモジュールに関する規約が定められている。ハードウェア依存部2002をこの規約にあわせて構築すれば、様々な計算機上でこのオペレーティングシステムを動作させることが可能になる。

【0181】この規約に従ったハードウェア依存の処理は、別オブジェクトファイルに分離され、カーネル本体とは切り離された領域2002にマップされている。カーネル本体2001とハードウェア依存部2002は、第一の実施の形態の場合と同様の外部参照機構により互いの公開モジュールを呼び出すことができ、見かけ上は1つのカーネルとして機能する。

【0182】このような場合は、カーネル本体のオブジェクトファイルを変更することなく、分離されたハードウェア依存の処理を実施するオブジェクトファイルの変更により第一の実施の形態、および、第二の実施の形態と同様の効果を得ることが可能である。

【0183】具体的には、分離されたオブジェクトファイルの処理において、物理メモリの割り当てが可能であること、割り込みレベル管理処理を変更できること、および、I/O資源の予約ができることが必要である。さらに、このオブジェクトファイル中にステップ1701から始まる割り込みハンドラと割り込みテーブル107を配置し、プロセッサの割り込みテーブルレジスタ104に登録する。そして、この分離されたオブジェクトファイルを共通領域203として第二のOSからも参照できるようにする。以上により、本発明の第一の実施の形態と同様の効果を得ることができる。

【0184】さらに、ハードウェア依存オブジェクトファイルが、第一のOSが停止したときに実行されるモジュールを持つ規定になっていれば、そのモジュールを変更すれば第一のOSの停止を検出でき、本発明の第二の実施の形態と同様の効果を得ることができる。

【0185】この実施の形態においては、カーネル本体を変更する必要がない。これにより、変更しなければならない部分が更に限定できる、カーネル本体を変更する

よりも容易に実施可能となる。

【0186】次に、本発明の第4の実施の形態について説明する。これまで説明してきた実施の形態では、共通領域203に配置していたのはサポートドライバや、ハードウェア依存オブジェクトファイルなどのオブジェクトファイルであった。しかし、本当に共通領域203に配置しなければならないモジュールとデータは、割り込みテーブル107、ステップ1701から始まる割り込みハンドラ、ステップ1601から始まるOS切り替え手順、および、図15に示したデータ構造分だけである。特に、第3の実施例のようにハードウェア依存部の処理を実施するオブジェクトファイル全体を共通領域203として第二のOSからも参照できるようにしてしまうと、第二のOSが誤って第一のOSのデータ構造にアクセスしてしまう可能性が高くなり問題である。

【0187】第4の実施の形態では、オブジェクトファイルの特定のセクションのみを共通領域203として第二のOSに見せる方法を提供する。この実施の形態では、オブジェクトファイルを生成するコンパイラが、命令コードとデータを配置するセクションをプログラム上で指定できる機能を持っている必要がある。

【0188】通常のオブジェクトファイルは、セクションとして命令コードを含むテキストセクションと、データを含むデータセクションを持っている。これに加えて、コンパイラの機能により共通領域203のためのセクションを追加する。さらに、オブジェクトファイルのヘッダ部分に格納されているセクションデータ809を参照して共通領域セクションのアドレス範囲を決定し、その部分だけを第二のOSに見せるようにページテーブルを構築すればよい。

【0189】ハードウェア依存処理をするモジュールを含むオブジェクトファイルを変更する場合を例として説明する。変更箇所のうち初期化に関連する部分、例えば、物理メモリの割り当て、I/O資源の予約、割り込みレベル管理部分の変更は第二のOSに見せる必要はない。第二のOSからも参照できなければならないのは、割り込みテーブル107、ステップ1701から始まる割り込みハンドラ、ステップ1601から始まるOS切り替え手順、および、図15に示したデータ構造分だけである。これらを、共通領域セクションに配置するようにプログラムを記述し、コンパイラの機能により共通領域セクションを生成する。

【0190】図21は、生成されたオブジェクトファイルの構成を示している。2100は、生成されたオブジェクトファイルを示す。オブジェクトファイル2100のヘッダ部の2101、ないし、2104は、オブジェクトファイル2100に含まれているセクションのデータを記述している。このうち2103と2104が、共通領域203用に新規に作成したセクションを表現するセクションデータである。対応するセクションは210

7と2108である。セクションデータ2103と2104の内容にしたがってセクション2107と2108のアドレスを求め、それらの領域だけを第二のOSのカーネル領域にマップするように第二のOSのページテーブルを構成すれば、ハードウェア依存オブジェクトファイル2100の他の部分を第二のOSから隠すことができる。

【0191】第4の実施の形態によれば、これまで説明した実施の形態よりも、更にOS間の独立性を高めることができ、OS間の干渉の少ない安全な計算機システムの構築が可能になる。

【0192】次に、本発明の第5の実施の形態について説明する。第5の実施の形態では、マルチプロセッサ構成の計算機で第二のOSを導入が可能になる。

【0193】図22は、本発明の第5の実施の形態での計算機装置を示す図である。2200は計算機装置である。計算機2200は2つのプロセッサ2201と2202、および、主記憶装置2203を持っている。また、第一の実施の形態と同様に、計算機起動プログラムを格納している記憶装置2204を持っている。

【0194】プロセッサ2201と2202について、プロセッサを起動したときと、初期化のための割り込みを受けた時とでは、制御を渡す物理アドレスが異なるものとする。

【0195】記憶装置2204に格納されている初期化割り込み処理プログラムは、予め定めた物理アドレスに格納されている値を物理アドレスとして、そのアドレスに制御を渡す。

【0196】また、バス2209を介して磁気ディスク装置2206、クロック割り込み生成装置2207、および、入出力装置2207等のデバイスが接続している。割り込みを発生するデバイスは、割り込み制御装置2205に接続し、更に、割り込みバス2211を介してプロセッサ2201と2202に接続している。各プロセッサは他のプロセッサに割り込みを送ることができるものとする。

【0197】割り込み制御装置2205について説明する。割り込み制御装置2205は、マルチプロセッサ構成のための機能を持っている。割り込み制御装置2205は、第一の実施の形態での割り込み制御装置112の割り込みマスク機能に加えて、それぞれのデバイスからの割り込みをどのプロセッサ、あるいは、プロセッサ群に通知するかを指定する機能を持っている。

【0198】図23は、割り込み制御装置2205の構成を示す図である。選択装置2301と割り込みマスクレジスタ2302の働きは、第1の実施の形態と同じである。それらに加えて、割り込み制御装置2205は、割り込み配送テーブル2310と、割り込み送信装置2305を持っている。

【0199】割り込み配送テーブル2310は、割り込

み制御装置2205に接続されたそれぞれのデバイスについて、どのプロセッサ、あるいは、プロセッサ群に割り込みを通知するかを示す値2311と、通知するときの割り込み番号2312を記録している。割り込み配送テーブル2302は、I/O命令により変更することができ、自由に設定可能である。

【0200】図23の例では、割り込み0と1はCPU0に、割り込み2はCPU1に配送するように設定されている。

【0201】割り込み送信装置2305は、選択装置2301からの信号を受けて、割り込み配送テーブル2310を参照して割り込み通知先と割り込み番号を決定する。そして、通知先と割り込み番号を表わす信号を割り込みバス2211へ送信する。

【0202】計算機2200は、起動するとプロセッサ2201だけが動作を開始するように構成されており、プロセッサ2201が記憶装置2204に格納されている起動プログラムを実行する。起動プログラムは、第一の実施の形態の場合と同様に磁気ディスク装置2206に格納されているカーネルローダを主記憶2203に読み込み実行する。カーネルローダは、パラメータテーブル1100を作成する。第5の実施の形態では、デバイスリストに計算機2200が何個のプロセッサを持っているかを示すデータが加えられる。

【0203】第一のOSのロード後、第一のOSの初期化処理を実行する。初期化の過程で、非ブートプロセッサ以外のプロセッサ用の初期化ルーチンのアドレスを予め定めた物理アドレスに格納し、プロセッサ2202に初期化割り込みを送る。プロセッサ2202は初期化割り込みを受けると、記憶装置2204に格納されているプログラムを実行し、非ブートプロセッサ初期化ルーチンに制御がわたる。非ブートプロセッサ初期化ルーチンは、ページテーブルレジスタや割り込みテーブルレジスタを設定して仮想アドレスモードに移行し、初期化処理を続ける。

【0204】本発明の第5の実施の形態では、図12のステップ1204の第二のOS用のデバイスの予約のときに、プロセッサも第二のOS専用であると予約する。ここでは、プロセッサ2202を予約するとして説明する。

【0205】マルチプロセッサ構成の場合、ステップ1201から始まる第一のOSの初期化手順のシステムデバイスの初期化で、非ブートプロセッサに初期化割り込みを送る。この場合、プロセッサ2201からプロセッサ2202に初期化割り込みが送られることになる。本発明では、予約されているプロセッサについては初期化割り込みを送らないことにする。したがって、カーネルの初期化がされてもプロセッサ2202はまだ動作していない。

【0206】また、ステップ1205のシステムデバイ

スの初期化では、割り込み制御装置 2205 の初期化も実施する。割り込み制御装置 2205 の初期化では、カーネル構成情報ファイル 700 の第二の OS の構成データ 704 を参照して、第二の OS が管理するデバイスの割り込みがプロセッサ 2202 に送られるように割り込み配送テーブル 2310 を設定する。

【0207】更に、図 14 のステップ 1401 から始まる第二の OS の初期化手順において、初期化ルーチンを第二の OS の初期化ルーチンのアドレスに設定して、ステップ 1407 でプロセッサ 2202 に初期化割り込みを送る。これにより、プロセッサ 2202 上で、第二の OS が走行を開始する。

【0208】第 1、ないし、第 4 の実施の形態と異なり、第二の OS の管理するデバイスの割り込みは、すべて割り込み制御装置 2205 により、第二の OS が動作しているプロセッサ 2202 へ送られる。このため、実行 OS を切り替える必要はなくなる。第一の OS はプロセッサ 2201 で動作し、第二の OS はプロセッサ 2202 で動作することになる。したがって、ステップ 1701 から始まる割り込み処理も不要になる。

【0209】第二の OS は、独自の割り込みテーブルをプロセッサ 2202 の割り込みテーブルレジスタに設定し、独自の割り込みハンドラを持てる。第一の OS の割り込みテーブルを変更する必要はない。但し、第一の OS が割り込み制御装置 2205 の割り込みマスクレジスタ 2302 を変更する場合は、第二の OS のデバイスからの割り込みをマスクしてしまわないように変更を加える必要がある。

【0210】第 5 の実施の形態においては、第 1 ないし 4 の実施の形態よりも、性能の良い計算機システムの構築が可能である。第 1 ないし 4 の実施の形態では、第一の OS は第二の OS がアイドルしている間のみ動作可能であったが、第 5 の実施の形態においては、第一の OS は、プロセッサは奪われてはいるが常に動作することが可能であり、同時に第二の OS も動作可能である。

【0211】更に、第 5 の実施の形態によれば、第一の OS と第二の OS とで共有しなければならない領域を小さくできる。第 1 ないし 4 の実施の形態では、共通領域 203 に、割り込みテーブル、割り込みハンドラや割り込み処理に付随するデータ構造、および、OS 切り替えコードを置かなければならなかった。第 5 の実施の形態では、これらはすべて必要なくなり、互いの OS が相手の OS を誤って破壊してしまう可能性を低くできる。

【0212】次に、本発明の第 6 の実施の形態について説明する。これまで説明した実施形態が 2 つの OS を同時実行する方式であるのに対し、第 6 の実施形態は 2 つ以上の複数の OS を同時実行する方式である。

【0213】第一の実施形態では、第二の OS が第一の OS よりも優先して実行するように制御するが、ここで説明する方式では、複数の OS 間に実行優先度を設定可

能である。例えば、割り込みについては、実行中の OS の優先度よりも低い優先度の OS が管理する割り込みの処理は延期される。実行中の OS の優先度よりも高い優先度の OS が管理する割り込みが発生した場合は、即座に実行 OS を切替え割り込み処理を開始する。

【0214】また、実行中の OS が自分よりの優先度の高い OS のモジュールを呼び出す場合は即座に実行 OS を切替えてモジュール呼び出しを実施する。その逆の場合、つまり、優先度の低い OS の側での処理が必要になる場合は、その OS が実行権を得るまで要求された処理を延期するように制御する。

【0215】図 24 は、本発明の第 6 の実施形態の計算機構成を示した図である。計算機構成はこれまでの実施例と同じであるが、主記憶装置 102 に複数の OS がロードされている様子を示している。各オペレーティングシステムは、ステップ 1401 に示した第二の OS をロードする手順と同じ手順でロードできる。

【0216】図 25 は、複数の OS の関係を概念的に示した図である。第一の実施の形態では 2 つの OS であったのに対し、ここでは第一の OS 以外に複数の OS が 1 つのプロセッサ上で動作していることを示している。

【0217】第一の OS の一部である共通領域が他の第 2、第 3、および、第 N の OS の論理空間 202、2503、および、2504 にマップされ全 OS から共通に利用できることを示している。各 OS の論理空間への共通領域のマッピングは、ステップ 1401 に示した手順により実施する。

【0218】更に、各 OS はそれぞれが管理する外部機器を持っていることを示している。第二の OS は機器 116 と 117、第三の OS は機器 2505 と 2506、第 N の OS は機器 2507 を管理することを示している。これらの機器を制御するための I/O アドレス範囲、および、割り込み番号は、図 7 に示すカーネル構成情報ファイル 700 に格納しておけば良い。図 7 では第二の OS 用の構成情報のみを格納しているように記述したが、他に第三、第四の OS の構成情報を格納しておく。ステップ 1201 からの初期化手順では、第二の OS だけでなく第一の OS 以外の全ての OS の資源を予約して、第一の OS が第一の OS 以外の OS が管理するデバイスへアクセスすることを禁止する。

【0219】ステップ 1201 から始まる手順では、ステップ 1202 で第二の OS 用の主記憶を確保している。これを、第一の OS 以外の複数の OS 用の主記憶を確保する処理とする。また、ステップ 1204 は第二の OS が管理するデバイスを予約する処理であるが、これを第一の OS 以外の OS が管理するデバイス資源を予約する処理とする。

【0220】本実施形態における第一の OS の初期化手順を図 26 に示す。ステップ 2602 と 2604 がステップ 1202 と 1204 に対応している。ステップ 26

04は、カーネル構成情報ファイル700を参照して第一OS以外のOSのデバイス資源を予約する。

【0221】この時、同時に割り込み管理テーブル1520の処理OS1521も設定する。また、各OSの構成情報には、デバイス資源の他にOSの優先度も記述する。

【0222】図27は、共通領域203に配置するデータ構造を示す図である。図15に示したデータ構造と比較して、割り込み識別テーブル1520、OS識別変数1530、および、遅延割り込み状態変数1540は同一のデータ構造である。OSコンテキストテーブル2710は、1510を拡張したデータ構造となっている。

【0223】テーブル2710は、各OSの実行を切替える時に必要になるデータを保存している。ページテーブル設定値2701とスタックポインタ設定2702は、あるOSを実行中に他のOSのモジュールを呼び出す時に設定するページテーブルとスタックポインタのアドレスを示している。また、ページテーブル保存値2703とスタックポインタ保存値2704は、優先度の高いOSへの切替を実施した時の優先度が低い方のOSのページテーブル値、および、スタックポインタ値を保存している。

【0224】実行状態2705は、それぞれのOSについて稼働中であるか、および、処理待ち中であることを示す値を格納する。ここで稼働中であるかとは、OSが起動されているかを示す。ある瞬間に実行中であることを示しているわけではない。各OSの起動処理は実行状態2705を設定する。

【0225】また処理待ち中であるとは、そのOSがアイドル状態でないことを示す。つまり、優先度の高いOSが実行中のため走行待ち状態にあることを示す。処理待ち中については、具体的にどの処理が待ちになっているかを記述してもよい。

【0226】優先度2706は、各OSの実行優先度を格納する。優先度は、第一OSの初期化処理手順のステップ2605で、構成ファイル700より読み出し設定する。

【0227】実行OSの切替え手順について、第二のOSが実行中であるとして説明する。また、OSの実行優先度が第一より第二、第二より第三の方が優先度が高く設定されているとする。第二のOSの処理により第三のOS上のプロセスが走行可能になったとする。この場合、第三のOS内部のプロセス起動モジュールを実行して第三のOS上のプロセスをスケジュールする。ここで、第三のOSは第二のOSよりも優先度が高いので、実行OSを即座に切替えて第三のOSのモジュールが実行される。切替え処理では現在のページテーブルアドレスとスタックポインタ値を、第二のOSのページテーブル保存値2703とスタックポインタ保存2704値に格納し、第三のOSのページテーブル設定値2701と

スタックポインタ設定値2702をページテーブルレジスタとスタックポインタに設定して実行OSを切替える。

【0228】第二のOSが第一のOSのモジュールを呼び出す場合、第一のOSの優先度は第二のOSの優先度よりも低いため、呼び出しは延期する。この場合、呼び出し要因を実行状態2705記録しておき、第一のOSが実行権を得た時に、すなわち、第二、および、第三のOSが処理を終了した時に、モジュール呼び出しを実施する。

【0229】図28は、実行OSの切替え手順を示すフローチャートである。ステップ2801からの処理の大部分は図16に示したステップ1601からの処理と同じである。この手順は、OSの優先度については考慮していないが、この手順を呼び出す前に呼出先OSの優先度と実行中OSの優先度を比較して、実際に呼び出して良い場合はステップ2801からの処理を実施すれば良い。

【0230】ステップ2801から始まる処理は、切替先OSと呼び出しモジュールのアドレスを引数として受ける。ステップ2801では、現在のページテーブルアドレスとスタックポインタを、OSコンテキストテーブル2710の現在実行中のOSのページテーブル保存値2703と、スタックポインタ保存値2704に保存する。現在実行中のOSは、OS識別変数1530により判定できる。

【0231】次のステップ2802では、OSコンテキストテーブル2710より切替先OSのページテーブル設定値2701とスタックポインタ設定値2702を取得し、ページテーブルとスタックの切替え処理を実施する。

【0232】ステップ2803では、遅延割り込み状態をクリアする。優先度が実行中のOS以上で、切替先未満のOSが管理するデバイスの遅延割り込み状態1540をクリアする。

【0233】例えば、第一のOSから第三のOSのモジュールを呼び出すとする。この場合、ステップ2803では第一のOSと第二のOSが管理するデバイスの遅延割り込み状態1540をクリアする。

【0234】続くステップ2804では、OSコンテキストテーブル2710の現在実行中のOSの実行状態2705を処理待ち中に設定し、ステップ2805でOS識別変数を切替先OSに設定し、ステップ2806で引数として渡されたモジュールを呼び出す。

【0235】切替先のOSは実施すべき処理がなくなった時点で、ステップ2807へ制御を戻す。ステップ2807では、OSコンテキストテーブルの各OSの実行状態2705と優先度2706を参照して、最も優先度が高い処理待ち中のOSを見つける。

【0236】ステップ2808は、ステップ2807で

選択したOSのコンテキストを、OSコンテキストテーブル2710のページテーブル保存値2703とスタックポインタ保存値2704より回復する。

【0237】次のステップ2809で、OS識別変数1530を選択したOSに設定する。

【0238】OSコンテキストテーブル2700の実行状態2705に延期されている処理があることが記録されている場合は、その延期されている処理を実行する（ステップ2810）。

【0239】続く処理では、遅延された割り込みを処理する。ステップ2812は、遅延割り込み状態1540を参照して、ステップ2807で選択されたOSが処理すべき割り込みが発生していないかを確認する。選択されたOSがどの割り込み番号の割り込みを管理しているかは、割り込み識別テーブル1520の処理OS1521を参照すれば分かる。

【0240】ステップ2812の検査の結果、処理待ちの割り込みが発生していると判定した場合はステップ2813へ進む。ステップ2813では、処理しなければならない割り込みが発生していることを選択されたOSが認識できるように選択されたOSのデータ構造を設定する。続くステップ2814は、割り込み識別テーブル1520を参照して、処理する割り込みの割り込みハンドラ1522を呼び出す。ハンドラの処理終了後、回復したコンテキストにしたがって、選択されたOSの実行を再開する。

【0241】処理待ち割り込みがなければ、そのまま回復したコンテキストにしたがって、選択されたOSの実行を再開する。

【0242】割り込み処理について説明する。図29は、割り込み処理手順を示すフローチャートである。図29に示す手順を実行するルーチンはプロセッサ103の割り込みハンドラとして割り込みテーブル400に登録される。また、全てのOSから参照可能な共通領域に配置される。

【0243】処理手順について説明する。まず、ステップ2901では、割り込み要因より割り込みを処理するOSを決定する。処理OSは、割り込み識別テーブル1520の処理OS1521に記録されており、これを参照して決定する。

【0244】割り込み発生時のOSと割り込み処理OSが同一ならば（ステップ2902）、割り込み識別テーブル1520に登録されている割り込みハンドラ1522を実行する（ステップ2903）。ハンドラ処理終了時に割り込まれた処理を再開する。

【0245】割り込み発生時のOSと割り込み処理OSが異なる場合は、2つのOSの優先度を比較する（ステップ2904）。実行中のOSの優先度の方が高い場合、ステップ2905へ進み、遅延割り込み状態変数1540を設定して、割り込まれた処理を再開する。

【0246】割り込み処理OSの優先度の方が実行中のOSよりも高い場合は、ステップ2906へ進み、割り込み処理を開始する。

【0247】ステップ2906では、割り込み識別テーブル1520より割り込みハンドラアドレス1522を取得する。続いて、割り込み発生時のコンテキストを保存し（ステップ2907）、割り込みハンドラを呼び出す（ステップ2908）。割り込みハンドラの呼び出しは、図28のステップ2801から始まる手順により実施する。

【0248】その後、割り込まれたOSに制御が戻った時に、割り込み発生時のコンテキストを回復し（ステップ2910）、割り込まれた処理を再開する。

【0249】次に、共通領域203に配置する処理モジュールについて述べる。共通領域203には、ステップ2801からのOS切替えモジュールと2901から始まる割り込みハンドラを配置する。この他の全てのOSから呼び出されるモジュールも共通領域203に配置する。例えば、各OSの割り込みハンドラを割り込み識別テーブル1520に登録するモジュールや、OSコンテキストテーブル2710の実行状態2705を設定するモジュールである。

【0250】各OSは、それぞれの初期化処理で共通領域203の登録モジュールを呼び出して割り込みハンドラを登録することにより、外部割り込みの処理が可能になる。また、何らかの処理により実行中のOSの優先度よりも低い優先度の他のOSの処理が可能になる場合、実行状態2705に処理待ち中であること、あるいは、処理待ちの要因を示す値を設定しなければならない。この場合も、共通領域203内の実行状態設定モジュールを呼び出すことにより設定する。

【0251】本実施形態によれば、第一のOSの他に複数のOSを1つのプロセッサで実行可能となり、特別な機能に特化したOSを組み合わせることが可能となる。例えば、リアルタイム性に優れたOSと計算機の信頼性を補完するOSとを組み合わせ、第一のOSの機能を補うことができる。各OSは独立しているのでさまざまなOSを組み合わせることができ、用途に応じた第一のOSの機能拡張が可能になる。更に、これまで説明した実施形態により得られる効果も損なわれることはない。

【0252】更に、OS間に優先度を設定できるので、リアルタイムOSの優先度を最も高く設定するといった設定も可能である。これにより、各OSの優れている機能を有効に利用することが可能になる。

【0253】本発明の第7の実施の形態について説明する。本実施形態は、第6の実施の形態の拡張である。第7の実施の形態では、複数動作している内のあるOSが障害停止した場合に、他の残ったOSが実行継続可能な制御方法を示す。

【0254】各OSの障害処理モジュールは、OSの実

行を停止する時に、OSコンテキストテーブル2710の実行状態2705を、停止中であると設定しなければならない。これは、前記実施形態の、共通領域203に配置した実行状態設定モジュールの呼び出しにより実施する。

【0255】また、各OSは、他のOSが実行を停止した時に呼び出されるOS停止通知ハンドラを持ち、割り込みハンドラと同様に共通領域203内のデータ構造に登録する。

【0256】次に、実行状態設定モジュールの処理について説明する。図30は、本モジュールの処理を示したフローチャートである。第三のOSが停止したとして説明する。

【0257】ステップ3001では、OSコンテキストテーブル2710の実行状態2705を設定する。

【0258】ステップ3002では、設定された実行状態を検査する。もし、OS実行停止に設定されたならば、ステップ3003へ進む。そうでなければモジュールの処理は終了する。

【0259】ステップ3003では、停止したOSよりも優先度が高い稼働中のOSのOS停止通知ハンドラを呼び出す。呼び出しは、ステップ2801からのOS切替えにより呼び出す。優先度が低いOSについては、実行状態2706にOS停止ハンドラ実行待ちであることを記録する。また、各OSのOS停止通知ハンドラは、割り込みハンドラと同様に、各OSの初期化時に共通領域内に配置されたデータ領域に登録されるものとする。

【0260】実行OS切替え処理では、切替先のOSが実行を停止していないかを検査する処理が必要である。図28のフローチャートにしたがって説明する。

【0261】まず、切替え処理を始める前に、切替先が稼働中かを検査しなければならない。ステップ2801の前に、OSコンテキストテーブル2710の実行状態2705を検査して、稼働中でなければ切替え処理を終了させる。

【0262】更に、ステップ2807での次の実行OSの選択では、処理待ち中のOSの検索対象から稼働していないOSを除外するようにする。

【0263】選択したOSの実行を再開する前に、選択したOSの実行状態2705を検査し、OS停止通知ハンドラの実行待ちが記録されているならば、OS停止通知ハンドラを実行する。

【0264】次に、割り込み処理について説明する。図29のフローチャートにしたがって説明する。ステップ2901の割り込み処理OS決定の後、処理OSが稼働中かを検査する。ここで稼働中でないと判定した場合、割り込みを解除して割り込まれた処理を再開させるようにする。

【0265】以上の処理により、第一のOS以外に複数のOSが1つのプロセッサ上で同時に動作している時に

いくつかのOSが障害により停止しても、他の残りのOSは動作を継続することが可能になる。

【0266】また、各OSの障害停止が他のOSに通知されるので、OS間で連携して処理を実施している場合でも、その通知をもとに各OSで障害処理を実施することが可能であり、計算機全体としての信頼性を高めることができる。

【0267】これまで述べた実施の形態では、それぞれのOSは互いに異なるOSであっても、同一のOSが含まれていても良い。

【0268】

【発明の効果】本発明は、特別なハードウェアの支援なしで、第一のOSに欠けている機能を容易に導入する方法であり、さらに、その機能は第一のOSとは全く独立して動作させることを可能にする。

【0269】本発明によれば、第一のOSに対する変更点は、オペレーティングシステムカーネルの初期化部分、デバイス資源予約、および、割り込み禁止制御部分に限定されるため、簡単に2つのオペレーティングシステムを同時に動作させることが可能になる。

【0270】仮想計算機による方式で発生するオーバーヘッドについて、本発明では、それぞれのデバイスについて、デバイスを管理するオペレーティングシステムを予め決定し、さらに、物理メモリについて利用できる範囲を初期化時に決定することにより、オペレーティングシステム同士が互いに干渉しないようにして、仮想計算機でのような複雑なソフトウェアによる制御を廃止し、命令エミュレーションによるオーバーヘッドも削減し、高速化のためのハードウェアも不要とした。

【0271】従来技術においても、第一のOSの構成要素として新たな機能をカーネルに追加することは可能であるが、第一のOSが停止してしまったとき、追加した機能モジュールも動作することはできない問題があった。本発明によれば、新たな機能を実現する構成要素を、第一のOSと独立して構成しておけば、第一のOSが停止してしまったとしても、その機能モジュールだけは継続して動作することが可能になる。信頼性が要求される機能モジュールを第二のOSとして組み込めば、第一のOSが停止してしまったときでも、何らかの回復処理を実現させるといったことが可能になる。

【0272】また、第一のOSは、第二のOSがアイドル時にしか動作できないことや、第二のOSの割り込みはいつでもすぐに処理することにより、第二のOSを優先させることができる。これにより、第一のOSが実時間処理に適合していなくても、第二のOSとして実時間処理に向くOSを導入すれば、第一のOSの特徴を活かしたままで、実時間処理性能に優れた計算機システムを構築することが可能になる。

【0273】また、第一のOSのカーネルが、カーネル本体とハードウェア依存の処理を実行するオブジェクト

ファイルとに分離されている場合、カーネル本体を変更しなくとも、後者のオブジェクトファイルを変更することで2つのオペレーティングシステムを同時に動作させることが可能になる。これにより、更に変更しなければならない部分が限定されるため、カーネル本体を変更するよりも容易に実施可能となる。

【0274】また、プログラムの記述によりオブジェクトファイル中に自由にセクションを設けることができるならば、それを利用して第一のOSと第二のOSとで共通に参照する必要のある領域を特別なセクションに閉じ込めることで、OS間の独立性を高めることができ、OS間の干渉の少ない安全な計算機システムの構築が可能になる。

【0275】本発明によれば、第一のOSに対する変更点は、オペレーティングシステムカーネルの初期化部分、デバイス資源予約、および、割り込み禁止制御部分に限定されるため、簡単に複数のオペレーティングシステムを同時に動作させることが可能になる。

【0276】複数のオペレーティングシステムを動作実行させた場合でも、上に示した効果を得ることができる。

【0277】従来技術においても、第一のOSの構成要素として新たな機能をカーネルに追加することは可能であるが、第一のOSが停止してしまったとき、追加した機能モジュールも動作することはできない問題があった。本発明によれば、新たな機能を実現する構成要素を、第一のOSと独立して構成しておけば、第一のOSが停止してしまったとしても、その機能モジュールだけは継続して動作することが可能になる。信頼性が要求される機能モジュールを第二、第三のOSとして組み込めば、第一のOSが停止してしまったときでも、何らかの回復処理を実現させるといったことが可能になる。

【0278】また、第一のOSの他に複数のOSを1つのプロセッサで実行可能なことにより、特別な機能に特化したOSを組み合わせることが可能となる。例えば、リアルタイム性に優れたOSと計算機の信頼性を補完するOSとを組み合わせ、第一のOSの機能を補うことができる。各OSは独立しているのでさまざまなOSを組み合わせることができ、他の効果を損なうことなく用途に応じた第一のOSの機能拡張が可能になる。

【0279】また、複数のOS間に実行優先度を設定できることにより、第一のOSが実時間処理に適合してなくても、実時間処理に向く第二のOSを最高優先度設定すれば、第一、および、第二のOS以外のOSの特徴を活かしたままで、実時間処理性能に優れた計算機システムを構築できることが可能となる。

【0280】また、第一のOSのカーネルが、カーネル本体とハードウェア依存の処理を実行するオブジェクトファイルとに分離されている場合、カーネル本体を変更しなくとも、後者のオブジェクトファイルを変更するこ

とで複数のオペレーティングシステムを同時に動作させることが可能になる。また、プログラムの記述によりオブジェクトファイル中に自由にセクションを設けることができるならば、それを利用して全てのOSで共通に参照する必要のある領域を特別なセクションに閉じ込めることで、OS間の独立性を高めることができ、OS間の干渉の少ない安全な計算機システムの構築が可能になる。

【図面の簡単な説明】

【図1】本発明の実施の形態の、計算機構成を示す図である。

【図2】本発明の実施の形態の、計算機構成を示す図である。

【図3】本発明の実施の形態の、ページテーブルの構成を示す図である。

【図4】本発明の実施の形態の、割り込みテーブルの構成を示す図である。

【図5】本発明の実施の形態の、割り込み制御装置の構成を示す図である。

【図6】本発明の実施の形態の、計算機のブート手順を示すフローチャートである。

【図7】本発明の実施の形態の、第一のOSのカーネル構成情報ファイルの構成を示す図である。

【図8】本発明の実施の形態の、オブジェクトファイルの構成を示す図である。

【図9】本発明の実施の形態の、オブジェクトファイルの構成を示す図である。

【図10】本発明の実施の形態の、オブジェクトファイルの構成を示す図である。

【図11】本発明の実施の形態の、カーネル起動パラメータテーブルのデータ構造を示す図である。

【図12】本発明の実施の形態の、第一のOSの初期化手順を示すフローチャートである。

【図13】本発明の実施の形態の、第一のOSのデバイス管理テーブルのデータ構造を示す図である。

【図14】本発明の実施の形態の、第二のOSの起動手順を示すフローチャートである。

【図15】本発明の実施の形態の、第一のOSと第二のOSが共有するデータ構造を示す図である。

【図16】本発明の実施の形態の、実行OSの切り替え手順を示すフローチャートである。

【図17】本発明の実施の形態の、割り込み処理手順を示すフローチャートである。

【図18】本発明の実施の形態の、第一のOSの割り込みマスク処理のためのデータ構造を示す図である。

【図19】本発明の第2の実施の形態の、第一のOSの障害停止処理を示すフローチャートである。

【図20】本発明の第3の実施の形態の、第一のOSと第二のOSのカーネル領域の構成を示す図である。

【図21】本発明の第4の実施の形態の、オブジェクト

ファイルの構成を示す図である。

【図 22】本発明の第 5 の実施の形態の、計算機システムの構成を示す図である。

【図 23】本発明の第 5 の実施の形態の、割り込み制御装置の構成を示す図である。

【図 24】本発明の第 6 の実施の形態の、計算機構成を示す図である。

【図 25】本発明の第 6 の実施の形態の、計算機構成を示す図である。

【図 26】本発明の第 6 の実施の形態の、第一の OS の初期化手順を示すフローチャートである。

【図 27】本発明の第 6 の実施の形態の、全ての OS が共有するデータ構造を示す図である。

【図 28】本発明の第 6 の実施の形態の、実行 OS の切

り替え手順を示すフローチャートである。

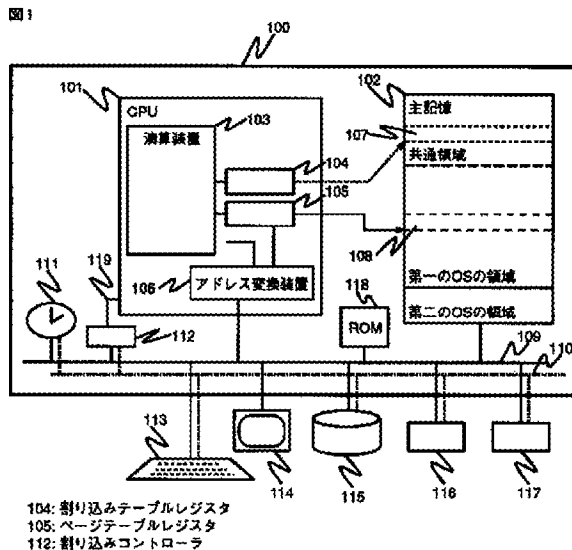
【図 29】本発明の第 6 の実施の形態の、割り込み処理手順を示すフローチャートである。

【図 30】本発明の第 6 の実施の形態の、OS 実行状態設定手順を示すフローチャートである。

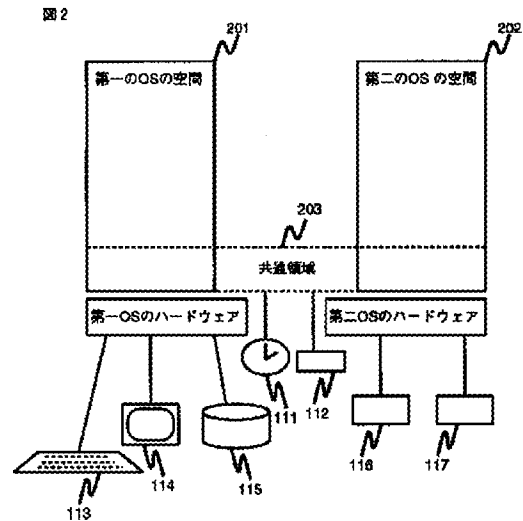
【符合の説明】

100 は計算機、101 はプロセッサ、102 は主記憶装置、103 は演算装置、104 は割り込みレジスタ、105 はページテーブルレジスタ、106 はアドレス変換装置、107 は割り込みテーブル、108 はページテーブル、109 はバス、110 は割り込み信号線、111 はクロック生成装置、112 は割り込み制御装置、113 ないし 117 は外部入出力装置、118 は割り込みバスである。

【図 1】



【図 2】



【図 3】

図 3

仮想ページ番号	有効	物理ページ番号
0	✓	0
1	✓	66
2	✓	23
3		
4		
5	✓	32
		!

300: ページテーブル

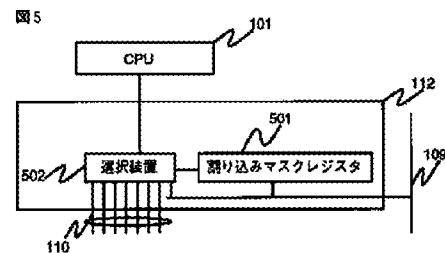
【図 4】

図 4

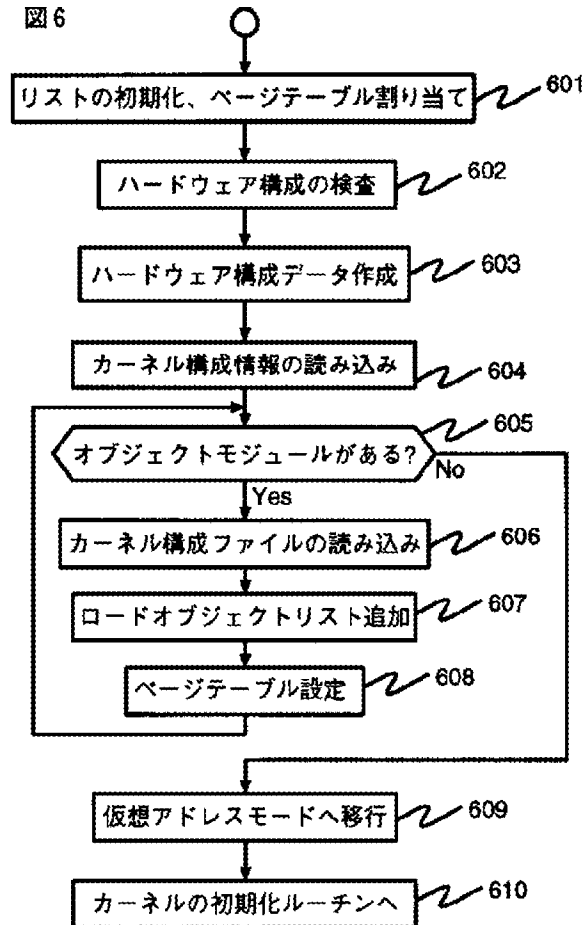
割り込み番号	割り込みハンドラ開始アドレス
0	800h
1	830h
2	500h
3	
4	
5	

400: 割り込みテーブル

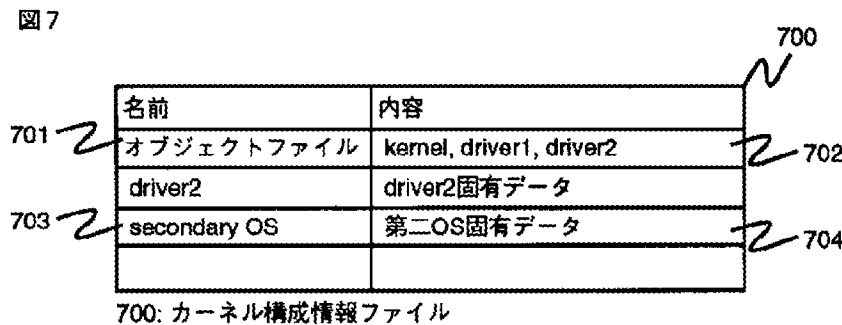
【図 5】



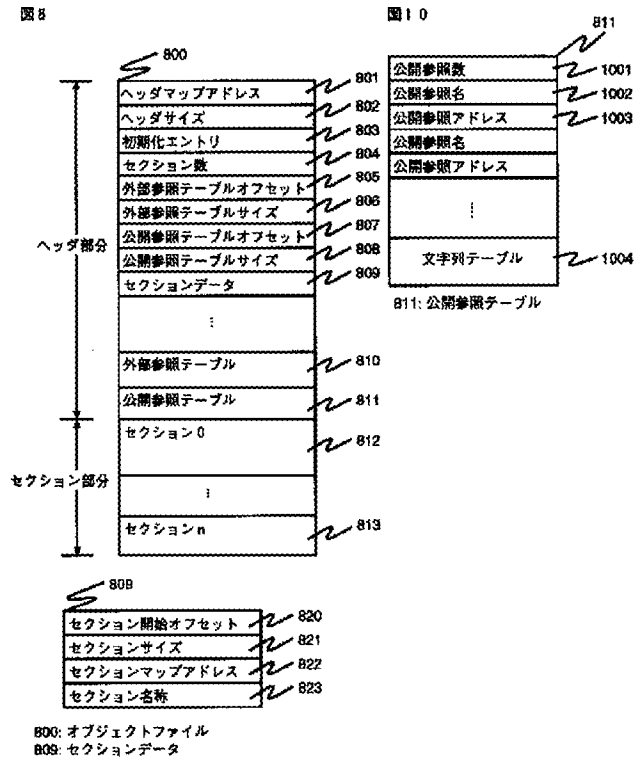
【図6】



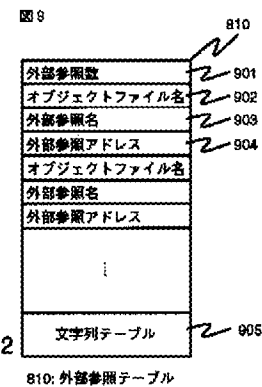
【図7】



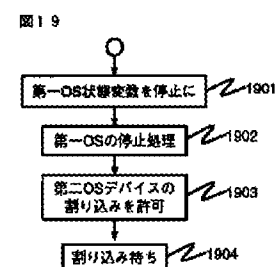
【図8】



【図9】

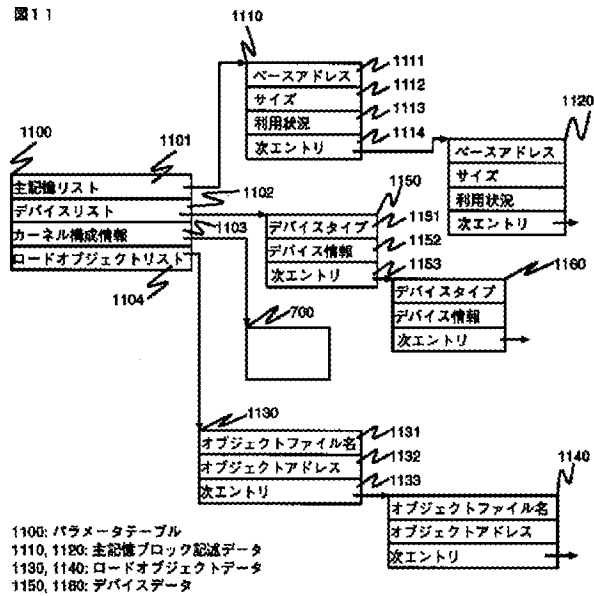


【図19】



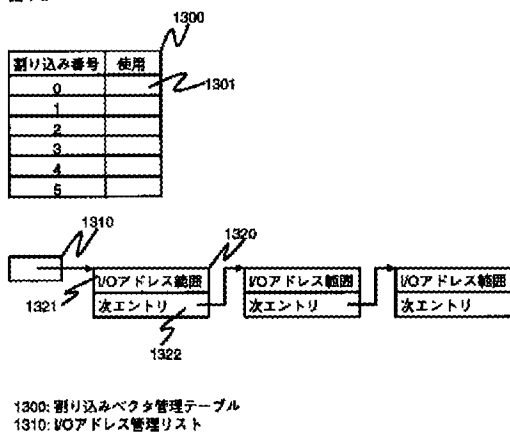
【図 11】

図 11



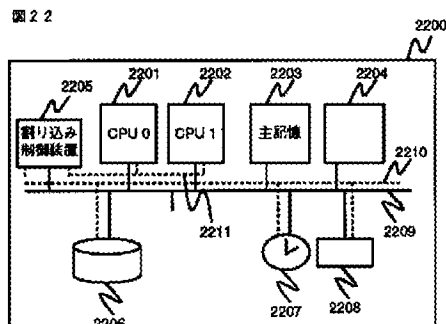
【図 13】

図 13



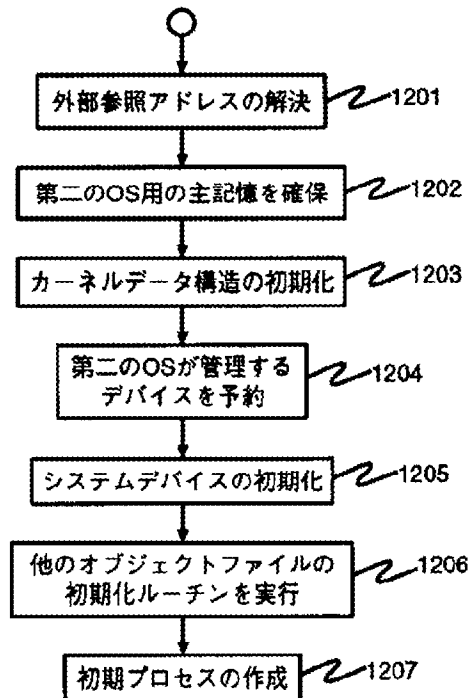
【図 22】

図 22



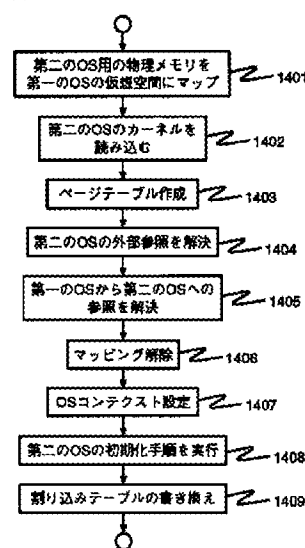
【図 12】

図 12



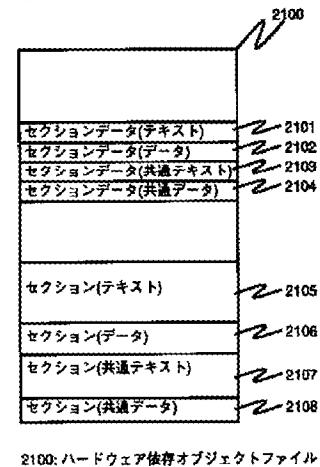
【図 14】

図 14



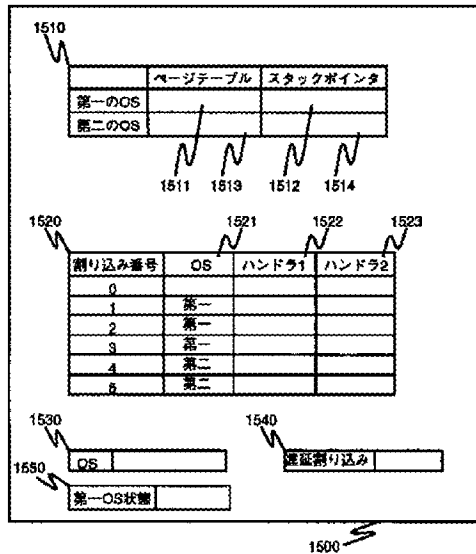
【図 21】

図 21



【図15】

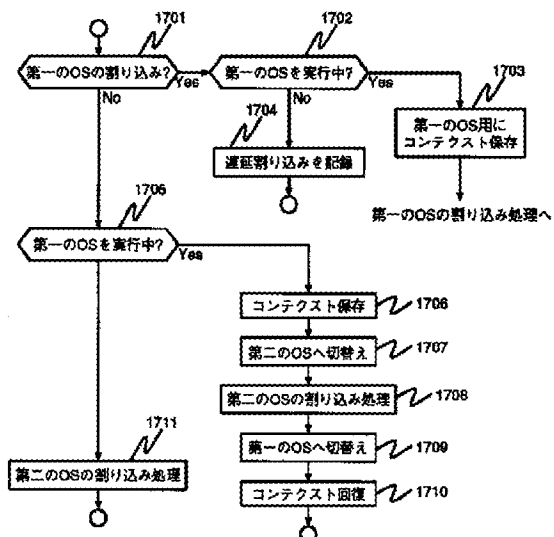
図15



1500: 共通領域
1510: OSコンテキストテーブル
1520: 割り込み識別テーブル
1530: OS識別変数
1540: 遅延割り込み状態変数
1550: 第一OS状態変数

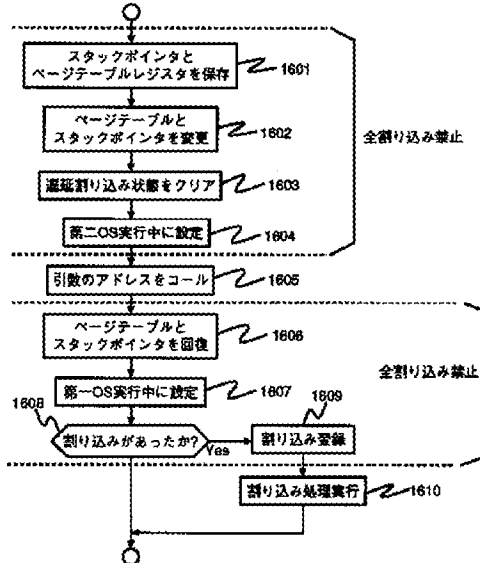
【図17】

図17



【図16】

図16



【図18】

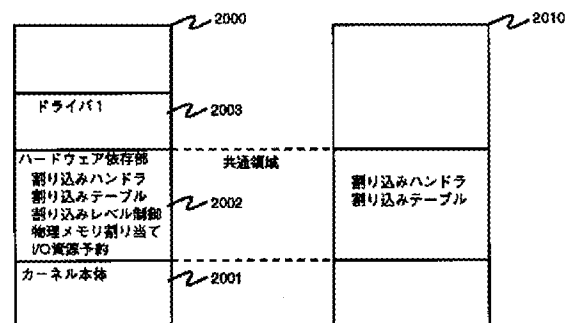
図18

割り込み禁止レベル	割り込み0	割り込み1	割り込み2	割り込み3	割り込み4	割り込み5	割り込み6
0							
1							
2							
3							
4							
5							
6							

1800: 割り込み禁止レベルテーブル

【図20】

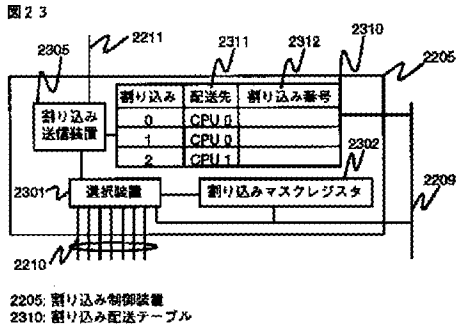
図20



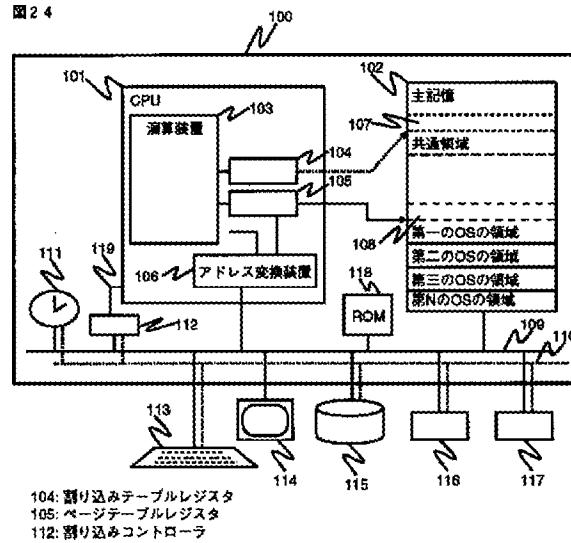
2000: 第一のOSのカーネル領域

2010: 第二のOSのカーネル領域

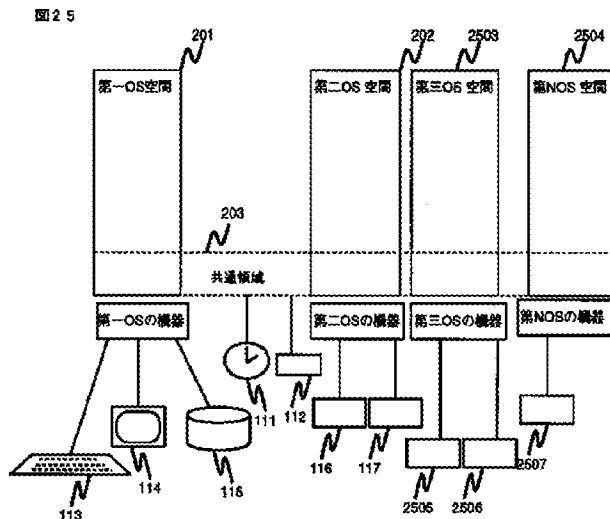
【図23】



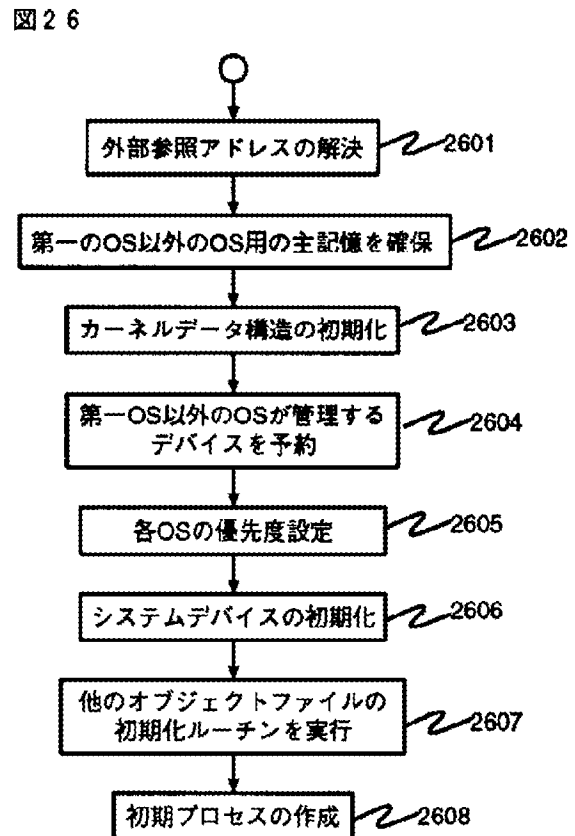
【図24】



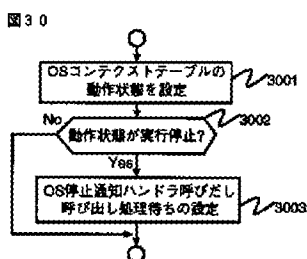
【図25】



【図26】

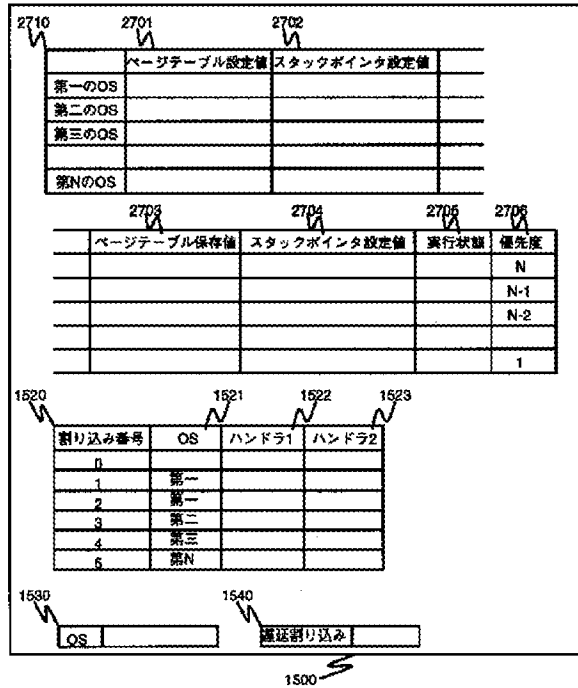


【図30】



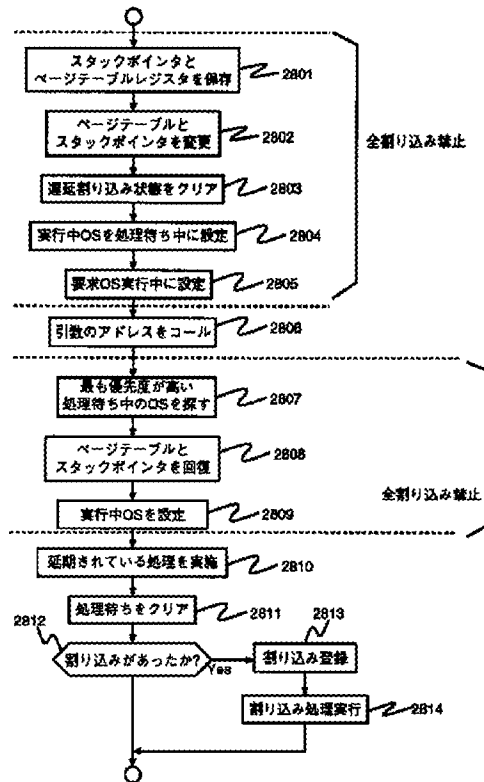
【図27】

図27



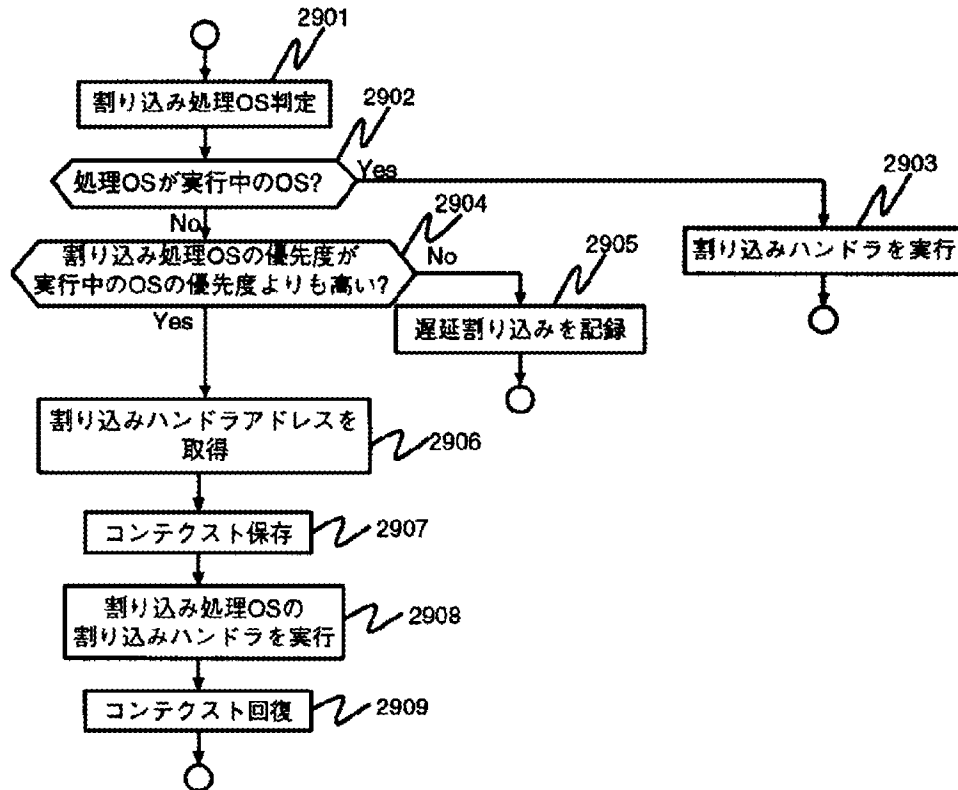
【図28】

図28



【図29】

図29



フロントページの続き

(72)発明者 大野 洋
茨城県日立市大みか町五丁目2番1号 株
式会社日立製作所大みか工場内

(72)発明者 井上 太郎
神奈川県川崎市麻生区王禅寺1099番地 株
式会社日立製作所システム開発研究所内
(72)発明者 柴田 隆
神奈川県横浜市戸塚区戸塚町5030番地 株
式会社日立製作所ソフトウェア開発本部内